'68' MICRO JOURNAL

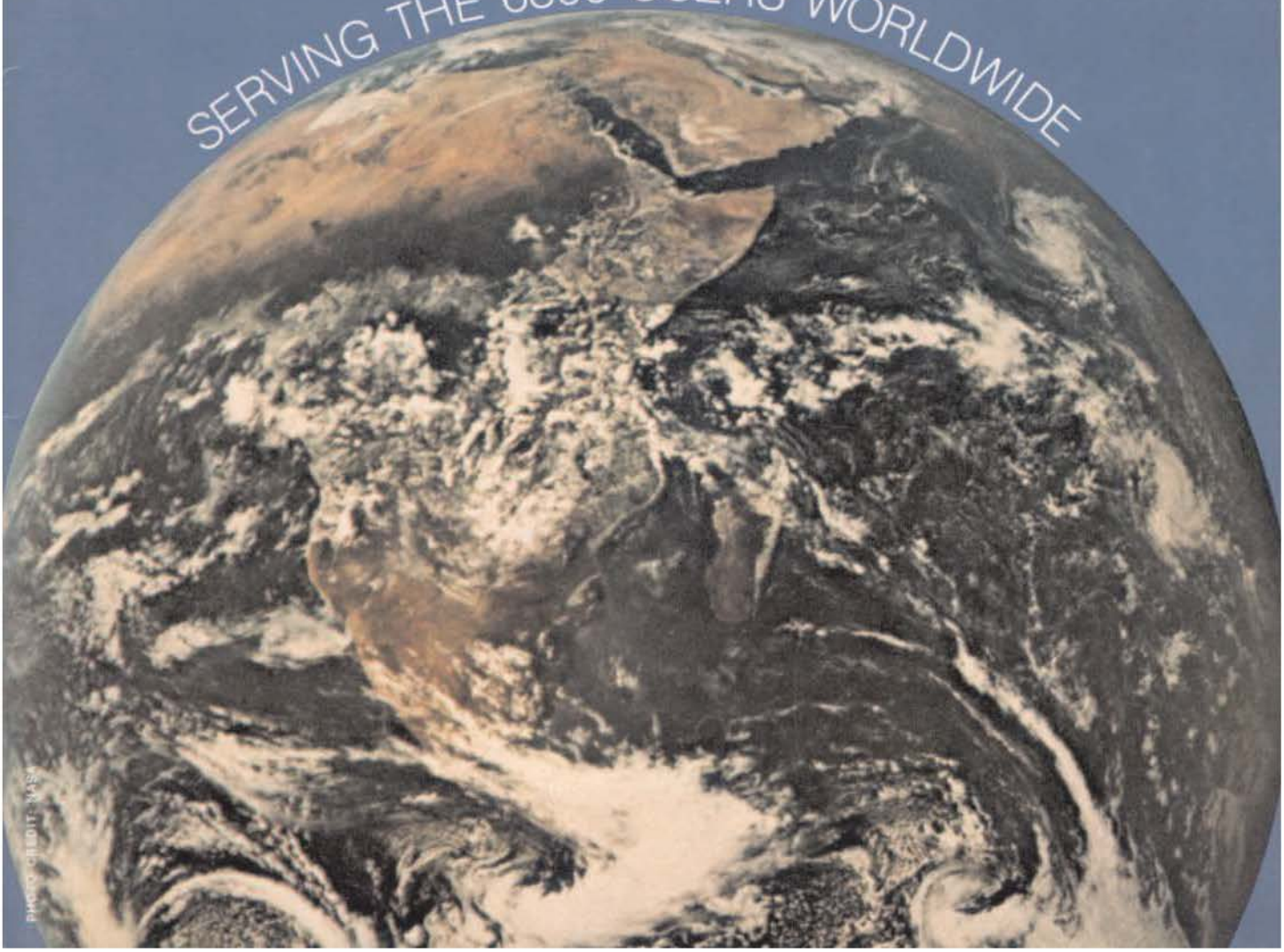
$2.50 USA

Australia	A $ 4.00	New Zealand	NZ $ 4.00
Singapore	S $ 8.00	Hong Kong	H $20.00
	Malaysia	M $ 8.00	

VOLUME II ISSUE 7 • Devoted to the 68XX User • July 1980

"Small Computers Doing Big Things."

SERVING THE 6800 USERS WORLDWIDE



SYSTEMS – SOLUTIONS

If you have a problem that can be solved by a computer—we have a systems solution.

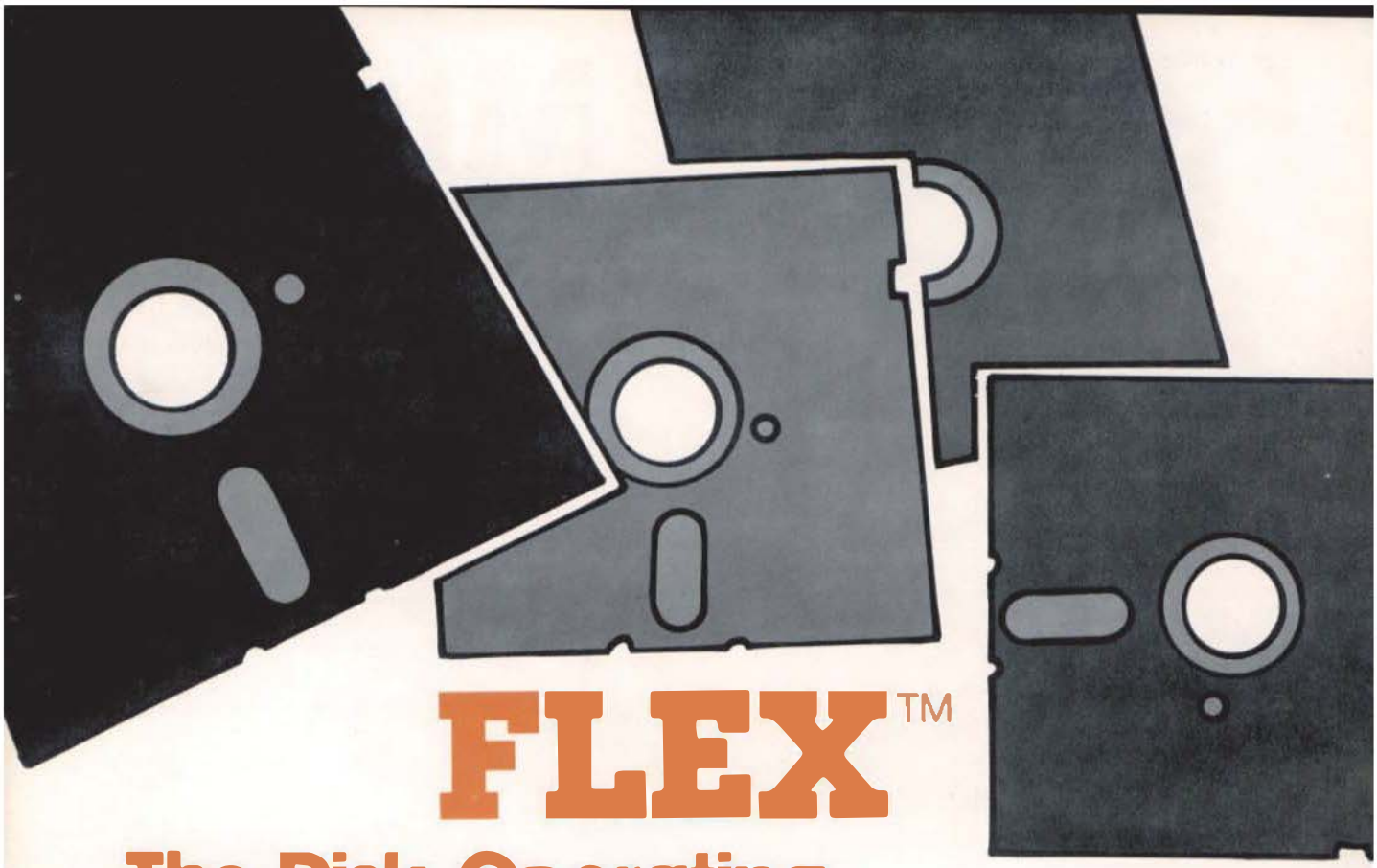
- Two central processors with maximum RAM capacities of 56K and 384 K bytes
- Three types of disk drives with capacities of 175K, 1.2M and 16M bytes
- Two dot matrix printers with 80 and 132 line capacity
- A Selectric typewriter interface and a daisy wheel printer

Match these to your exact need, add one or more of our intelligent terminals and put together a system from one source with guaranteed compatibility in both software and hardware.

Southwest Technical Products systems give you unmatched power, speed and versatility. They are packaged in custom designed woodgrain finished cabinets. Factory service and support on the entire system and local service is available in many cities.

SWTPC

SOUTHWEST TECHNICAL PRODUCTS CORPORATION
219 W. RHAPSODY
SAN ANTONIO, TEXAS 78216 (512) 344-0241



FLEX™

The Disk Operating System For 6800 and 6809 Users

FLEX™ is the most widely used disk operating system for the 6800 and 6809 microprocessors. Field proven for over two years, it has become an industry standard. FLEX is unparalled in the amount of 6800/6809 support software being marketed. Two new versions are now available and each includes a disk editor and assembler:

FLEX for the EXORciser™ **$150.00**
Runs on a Motorola EXORciser with EXORdisk™ II or III. Requires no hardware modifications with the possible exception of memory re-addressing. Uses the some boot as MDOS™.

FLEX for General Use **$150.00**
Fully documented to allow a user to write his own terminal and disk I/O routines to adapt to most any hardware. Three system requirements are: (1) at least 12K of RAM at $0000; (2) 8K of RAM at $A000 for 6800 or $C000 for 6809; (3) floppy disk drive capable of 256-byte, soft sectors. This package is not for beginners!

FLEX Support Software

Extended BASIC	$100.00
Standard BASIC	65.00
6809 Diagnostics Package	75.00
Text Processing System	60.00
Sort/Merge	75.00
68000 Cross Assembler	250.00
6809 Cross Assembler	100.00
6809 FLEX Utilities	60.00
6800 FLEX Utilities	100.00
6809 Debug Package	75.00
6800 Debug Package	55.00
FLEX for SWTPc	90.00

Be sure to specify disk size and 6800 or 6809. All orders should include 3% postage and handling (10% on foreign orders). Mastercharge and Visa are welcomed. Write for a complete software catalog.

technical systems consultants, inc.
Box 2570, West Lafayette, IN 47906
(317) 463-2502 Telex 276143

'68' MICRO JOURNAL

Portions of text prepared using the following.

SWTPC 6800-6809-DMAF2-CDS1-CT82-Sprint 3
Southwest Technical Products
219 W. Rhapsody
San Antonio, Texas 78216

EDITOR - WORD PROCESSOR
Technical Systems Consultants, Inc.
Box 2573, W. Lafayette, IN 47906
FLEX is TM of TSC

GIMIX Super Mainframe-Assorted memory boards
GIMIX Inc.
1337 West 37th Place
Chicago, Il 60609

Publisher: Don Williams Sr.

Executive Editor: Larry Williams

Subscriptions and Office manager
Mary Robertson

General Girl 'Friday'
Joyce Williams

Contributing Editors:

Dr. Jack Bryant
Dr. Chuck Adams
Dr. Theo Elbert
Dr. Jeffery Brownstein
Dale Puckett
Russell Gore
Ron Anderson
John Jordon

Typography and color work:
Williams Inc.
Chattanooga, TN 37421

* CONTENTS *

Send All Correspondence To:

'68' Micro Journal
3018 Hamill Rd.
PO Box 849
Hixson, Tennessee 37343

— Phone —
Office: 615-870-1993
Plant: 615-892-7544

'68' Micro Journal is published 12 times a year by '68' Micro Journal, 6131 Airways Blvd., Chattanooga, TN 37421. Second Class postage paid at Chattanooga, TN. Postmaster: Send Form 3579 to '68' Micro Journal, PO Box 849, Hixson, TN 37343.

1-Year $14.50 2 Years $26.00 3 Years $36.50

—ITEMS SUBMITTED FOR PUBLICATION—

(Letters to the Editor for Publication) All 'letters to the Editor' should be substantiated by facts. Opinions should be indicated as such. All letters must be signed. We are interested in receiving letters that will benefit or alert our readers. Praise as well as gripes is always good subject matter. Your name may be withheld upon request. If you have had a good experience with a 6800 vendor please put it in a letter. If the experience was bad put that in a letter also. Remember, if you tell us who they are then it is only fair that your name 'not' be withheld. This means that all letters published, of a critical nature, cannot have a name withheld. We will attempt to publish 'verbatim' letters that are composed using 'good taste.' We reserve the right to define (for '68' Micro) what constitutes 'good taste.'

(Articles and items submitted for publication) Please, always include your full name, address, and telephone number. Date and number all sheets. TYPE them if you can, poorly handwritten copy is sometimes the difference between go, no-go. All items should be on 8X11 inch, white paper. Most all art work will be reproduced photographically, this includes all listings, diagrams and other non-text material. All typewritten copy should be done with a NEW RIBBON. All hand drawn art should be black on white paper. Please no hand written code items over 50 bytes. Neatly typed copy will be directly reproduced. Column width should be 3¼ inches.

(Advertising) Any Classified: Maximum 20 words. All single letters and/or numbers will be considered one (1) word. No Commercial or Business Type Classified advertising. Classified ads will be published in our standard format. Classified ads $7.50 one time run, paid in advance.

Commercial and/or Business advertisers please write or phone for current rate sheet and publication lag time.

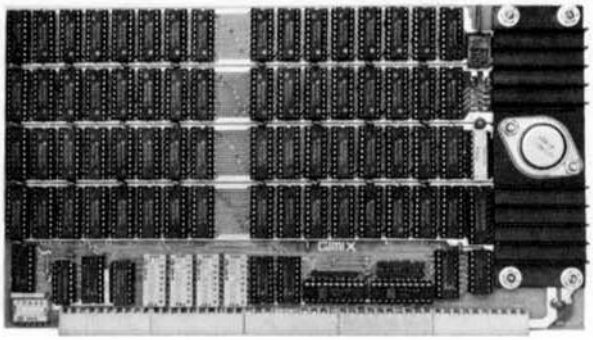
LOOK

WHAT'S COOKING on the FIFTY BUS

32K STATIC RAM BOARDS

Designed for use with:

★ **Existing SS50 Systems** ★ **SS50C Extended Address Systems**



- Assembled
- Burned In
- Tested

16K . . . $328.12
24K . . . $438.14
32K . . . $548.15

16K and 24K Versions are sockeled for 32K and require only additional 2114's for expansion.

FEATURES:

- Decoding for 4 Extended Address Lines (allows memory decoding up to 1 megabyte)
- DIP-switch to set extended addressing or disable it
- 4 separate 8K blocks, addressable to any 8K boundary by DIP-switch
- Each 8K block may be individually disabled
- Write protect either of two 16K sections
- Low power consumption — uses 2114L low power RAMS
- Fully Sockeled
- Gold Bus Connectors
- Guaranteed 2MHz operation

AND NOW . . . GIMIX OFFERS YOU A Choice of 6800 or 6809 CPU CARDS

You can order your system to fit your needs or select one of the below featured systems. Please contact the factory for further information and availability.

Add as much memory as you need using GIMIX Static RAM Cards for the utmost in reliability.

32K 6800 SYSTEM $1,694.59
Includes: Chassis, 6800 CPU, 32K RAM BOARD, I/O card

32K 6809 SYSTEM $1,844.69
Includes: Chassis, 6809 CPU, 32K RAM BOARD, I/O card

32K 6809 PLUS SYSTEM $1,994.79
Includes: Chassis, 32K RAM BOARD, I/O Card, and features our 6809 PLUS CPU Card with the Time of Day Clock option with battery back-up installed, as well as the 6840 Timer Package that provides 3 independent 16 bit counters.

This system also allows the following options to be added at additional cost:

- Battery back-up of the 1K RAM by substituting CMOS parts.
- A 9511 or 9512 Arithmetic Processor.
- GIMIX or SWTP Dynamic Address Translators.

EXPORT NOTES:

For 50Hz 230V C.V. POWER SUPPLY . . . . . Add $30.00
80 x 24 VIDEO BOARDS — Specify Format (No Added Charge)

On Orders under $250.00 for a Single Board, or Chips, please Add $30.00 Handling and we will ship Air Mail Prepaid. On all other orders we will ship via Emery Air Freight Collect, and we will charge no handling. All orders must be prepaid in U.S. Funds. Please note that foreign checks have been taking about eight weeks for collection, so we would advise wiring money or checks drawn on a bank account in the U.S. Our bank is the Continental Illinois National Bank of Chicago, Account #73-32033. Visa or Master Charge also accepted.

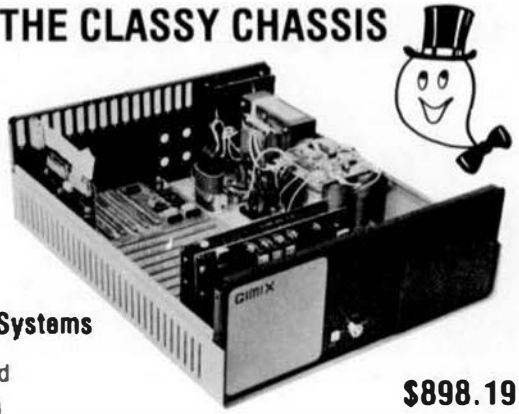
FACTORY PRIME STATIC RAMS

2114L 450 ns . . $5.90 300 ns . . $6.40 200 ns . . $6.90
4044 450 ns . . $5.90 250 ns . . $6.90

ADD $5.00 HANDLING ON ORDERS UNDER $200.00

GIMIX® and GHOST® are Registered Trademarks of GIMIX INC.

THE CLASSY CHASSIS



$898.19

- 25 amp (5V) ferro-resonant constant voltage power supply.
- Heavy weight aluminum cabinet with 3 position key switch, fan, and provisions for two 5" disk drives;
- 6800/6809 Mother Board, fifteen 50 pin and eight DIP-switch addressable 30 pin slots (gold plated pins), fully decoded;
- Baud rate generator on I/O section of Mother Board.

I/O BOARDS

for the 30 PIN BUS:

1 Port Serial (RS 232 or 20MA, current loop)	$ 88.41
2 Port RS 232 Serial	128.43
2 Port Parallel	88.42

for the 50 PIN BUS:

6 Port RS 232 Serial	288.40
6 Port RS 232 Serial with on board Baud Rate generator.	318.46
6 Port Parallel	198.45

BOTH 6809 SYSTEMS FEATURE OUR NEW TERMINAL BASED GMXBUG 09 SYSTEM MONITOR

GMXBUG 09 includes advanced debugging tools, utility, and memory manipulation routines.

Both 6809 Systems:

- ★ Can be reconfigured to allow use of other system monitors (OS-9 and SBUG-E)
- ★ Include 1K of Scratchpad RAM on the CPU
- ★ Allow optional software switching of system monitors.

2MHz 6809's at slight additional cost when they become available.

Phone, write, or see your dealer for details and prices on our broad range of Boards and Systems for the SS50/SS50C bus and our AC Power Control Products for all computers.

GIMIX INC.

The Company that delivers

Quality Electronic products since 1975.

1337 WEST 37th PLACE, CHICAGO, IL 60609
(312) 927-5510 • TWX 910-221-4055

SEE GHOST ADS PAGES 36, 44 & 48

SMOKE SIGNAL BROADCASTING

Presents

3 Powerful New SS-50/SS-50C Boards

DCB-4
Disk Master
Double Density Controller Board and DOS68D Double Density DOS
$449.00

The new DCB-4 is a truly state-of-the-art development which allows up to 366K bytes to be stored on a single 5¼" disk and has these outstanding features:

- Up to four 5¼"and four 8" drives can be handled in the same system with a user definable logical unit table. (DOS68D will be compatible with future hard disk systems).
- Under software control, the user can select the following for any drive:
  - ☆ Single sided or double sided operation.
  - ☆ Single density or double density data.
  - ☆ 5¼" or8".
  - ☆ Stepping Rate.
  - ☆ 40 track or 35 track density on double sided 5¼" drives.
  - ☆ User can select the system boot configuration.
- Occupies only 16 bytes of memory space (F760-F76F standard). User selectable to any 16 byte address space.
- Can read and write a single sector by itself. On-board buffer memory allows full interrupt capability in interrupt driven systems. Once data transfer has been initiated, no more processor time is required.
- Contains extended decoding circuitry for extended addressing per SS–50C bus which can be enabled by an option jumper.
- SSB provides a means for copying software written by older versions of DOS68 to be read by DOS68D. All new media formatted by DOS68D can be read by all older versions of DOS68. DOS68 is SSB's 6800 disk operating system.
- Track 0 of side 0 is recorded in single density per IBM standard.
- Phase-locked-loop assures highest data integrity attainable.

All of these features are available for immediate delivery on one standard 5¼" x 9" 50 pin SS–50/SS–50C card for only $449.00. The price includes DOS68D version 5.1, MONITOR object code on diskette, and a manual with the source listing.

SCB-69
Super Computer Board
6809CPU Board
$299.00

The most versatile 6809 CPU Board on the market is now available from Smoke Signal Broadcasting and has the following features:

- Standard 2 MHz operation. (Shipping 1.5 MHz until August 80)
- 20 bit address generation for up to 1 Mbyte of memory. Uses an improved address translation RAM which is compatible with present extended addressing schemes yet requires much less overhead when used in multi-user systems.
- All on-board devices can be switch selected to occupy any or all extended pages. Any on-board device may be disabled and its memory space is then available for external memory.
- Standard real–time clock (time–of–day, day–of–week, day–of–month) with battery back up capable of generating programmable interrupts.
- Up to 20K of EPROM can be installed on the CPU Board.
- Standard 1K of RAM on board.
- Includes improved 6809 Monitor (and source listing).
- Contains an FPLA for decoding EPROM address and optional devices. Switches are used to select 2K/4K EPROM and Fast/Slow I/O.
- Contains provision for optional 9511/9512 floating point processor.
- NMI line is user selectable to work with either SS–50 or SS–50C busses.

Price for the new SCB–69 is only $299.00 for an assembled, burned-in fully tested board.

M-32-X
32K
Memory Board
$539.00 $439.00

The first and only 32K Static Ram Board on standard size (5½" x 9") SS–50/SS–50C Bus Circuit Card is made by Smoke Signal.

- Switch selectable to any 4K boundary.
- Any 4K block may be switch enabled or disabled.
- Fully compatible with SS–50C extended addressing (allows memory decoding up to 1 Mbyte).
- Extended addressing capability may be switched off for compatibility with SS–50 systems.
- Gold Bus Connectors for high reliability.
- Guaranteed 2MHz operation (tested at 2.2 MHz).
- Low power consumption – 8 volts at 2.4 amps typical.

M–32–X 32K Memory Board is priced at $539.00.

M–24–X 24K Memory Board expandable to 32K, is $439.00.

And our M–16–X 16K board is back to the old price of $299.00.

SMOKE SIGNAL BROADCASTING®

31336 Via Colinas, Westlake Village, CA 91361, (213) 889-9340

SOFTWARE...

MSI has MORE!

WE INVITE YOU TO LOOK AT OUR NEW SOFTWARE CATALOG WHICH OFFERS NEW PROGRAMS FOR YOUR 6800 SYSTEM.

•All FLEX™ Programs from TSC are now available for MSI Computer Systems.

•MULTI-DISK FLEX™ from MSI allows the use of any combination of MSI disk devices to be used simultaneously, including the HD-8/A 10 megabyte drive.

•SORT/MERGE Program can be used manually or within other BASIC or assembler programs to perform high speed sorts of data files.

•Hemenway Associates Software Products for use under FLEX™ are available on the MSI System.

•TRS-80/MICROSOFT BASIC - MSI BASIC Translator allows MSI users to run the large library of basic programs written for the TRS-80 and other similar systems.

•SOFTWARE LIBRARY Programs keep track of all diskette and hard disk directories, giving alphabetical listings of available programs.

•SDOS Operating System.

•MULTI-USER/MULTI-TASKING SDOS Operating System allows any user to perform edits, assemblies, compilations, or program executions independently and simultaneously.

•All MSI software is supported on four (4) disk systems: quad density minifloppy, single and double density 8" floppy, as well hard disk systems.

•Complete BUSINESS APPLICATION PACKAGES including sales order entry, accounts receivable, inventory management, purchase order entry, accounts payable, and general ledger are available on MSI hard disk systems.

•PLOTTING PACKAGE gives daisy-wheel printers the capacity to perform graphics operations.

•LETTERWRITER Word Processing Software allows the use of daisy-wheel printers to generate documents and to handle correspondence automatically.

FLEX™ is a registered trademark of Technical Systems Consultants, Inc.

Send for your catalog today.

Midwest Scientific Instruments

220 W. Cedar • Olathe, Kansas 66061 • 913-764-3273

TWX 910 749 6403 (MSI OLAT)

Telex 42525 (MSI A OLAT)

A Few Extraordinary Products for Your 6800/6809 Computer

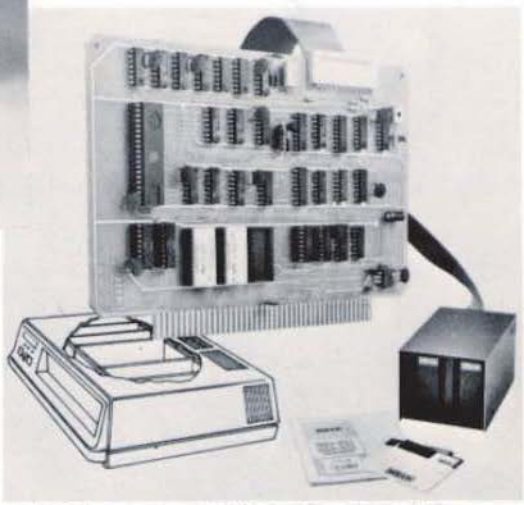
SS-50 Bus LFD-400™ and LFD-800™ Systems

From Percom . . .

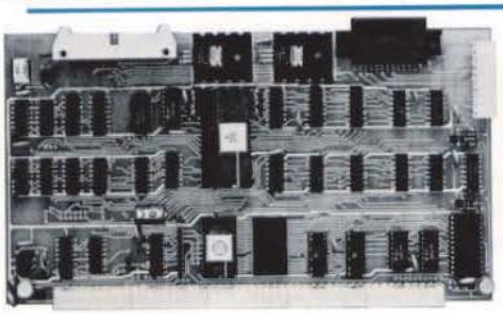
Low Cost Mini-Disk Storage in the Size You Want

Percom mini-disk systems start as low as $599.95, ready to plug in and run. You can't get better quality or a broader selection of disk software from any other microcomputer disk system manufacturer — at any price!

Features: 1-, 2- and 3-drive systems in 40- and 77-track versions store 102K- to 591K-bytes of random access data on-line • controllers include explicit clock/data separation circuit, motor inactivity time-out circuit, buffered control lines and other mature design concepts • ROM DOS included with SS-50 bus version — optional DOSs for EXORciser® bus • extra PROM sockets on-board • EXORciser® bus version has 1K-byte RAM - supported by extended disk operating systems; assemblers and other program development/debugging aids; BASIC, FORTRAN, Pascal and SPL/M languages; and, business application programs.



EXORciser® Bus LFD-400EX™ -800EX™ Systems



Versatile Mother Board, Full-Feature Prototyping Boards

Printed wiring is easily soldered tin-lead plating. Substrates are glass-epoxy. Prototyping cards provide for power regulators and distribu ed capaci or bypassing, accommodate 14-, 16-, 24- and 40-pin DIP sockets. Prototyping boards include bus connectors, other connectors and sockets are optional.

MOTHERBOARD — accommodates five SS-50 bus cards, and may itself be plugged into an SS-50 bus. Features wide-trace conductors. Price: $21.95

SS-50 BUS CARD — accommodates 34- and 50-pin ribbon connectors on top edge, 10-pin Molex connector on side edge. Price: $24.95.

SS-30 BUS CARD — 1¼-inch higher than SWTP I/O card, accommodates 34-pin ribbon connector and 12-pin Molex connec or on top edge. Price: $14.95.

The SBC/9™, A "10" By Any Measure.

The Percom SBC/9™ is an SS-50 bus compatible, s and-alone Single-Board Computer. Configured for the 6809 microprocessor, the SBC/9™ also accommodates a 6802 without any modification. You can have state-of-the-art capability of the '09. Or put to work the enormous selection of 6800-coded programs that run on the '02.

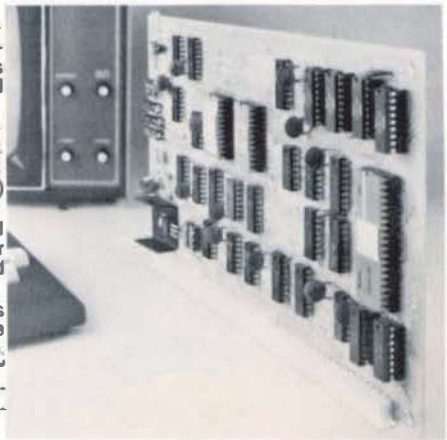
The SBC/9™ includes PSYMON™, an easily ex ended 1-Kbyte ROM OS. Other features include:

- Total compatibility with the SS-50 bus. Requires no changes to the motherboard, memory or I/O.
- Serial port includes bit-rate generator. RS-232-C compatible with optional subminiature 'D' connector installed. 10-pin Molex connector provided.
- Eight-bit, non-latched, bidirectional parallel port is multi-address extension of system bus. Spans a 30-address field; accommodates an exceptional variety of peripheral devices. Connector is optional.
- Includes 1-Kbyte of static RAM.
- Costs only $199.95 with PS ON™ and comprehensive users manual that includes source listing of PSYMON™.

The Electric Window™: Instant, Real-Time Video Display Control

Memory residency and outstanding software control of display format and characters make this SS-50 bus VDC card an exceptional value at only $249.95. Other features:

- Generates 128 characters including all ASCII displayable characters plus selected Greek letters and other special symbols.
- Well-formed, easy-to-read 7x12-dot characters. True baseline descenders.
- Character-store (display) memory included on card.
- Provision for optional character generator EPROM for user defined symbols.
- Comprehensive users manual includes source listing of Driver software. Driver — called WINDEX™ — is also available on mini-diskette through the Percom Users Group.



™ trademark of Percom Data Company, Inc.
® trademark of the Motorola Corporation.
Prices and specifications subject to change without notice.

PERCOM

PERCOM DATA COMPANY, INC.
211 N. KIRBY GARLAND, TEXAS 75042
(214) 272-3421

Products are available at Percom dealers nationwide. Call toll-free, 1-800-527-1592, for the address of your nearest dealer, or to order direct.

TRAPDOOR FUNCTION ENCRYPTION WITH THE 6800

T. F. Elbert and R. Enzian
The University of West Florida
Pensacola, Florida 32504

Cryptography, so long considered to lie within the almost exclusive purview of the military and intelligence communities, has sprung upon the commercial computer scene with amazing alacrity, and has even now begun its entry into the home computer market. The National Bureau of Standards, anticipating the need within the federal government for encryption of nonclassified data, has promulgated its Data Encryption Standard (DES) as the single method to be used within the federal establishment (reference 1). With such innovations as electronic mail and electronic funds transfer soon to become a practical reality on a large scale, data encryption will undoubtedly become a commonplace feature of digital data transmission.

Technically speaking, encryption of digital data transmission involves use of a cipher, in which a fixed relationship exists between the number of characters in the plaintext and its ciphertext transformation, as distinguished from a code, in which no such fixed relationship exists. Ciphers have been in use for centuries, one of the simplest being termed the "Caesar cipher" because its use can be traced back to Julius Caesar. While early ciphers were effective, modern science and its accompanying technology have made virtually every cipher susceptible to cryptanalysis using correlation and statistical methods. The sole exception, generally conceded to be absolutely secure, is the Vernam cipher. This cipher, known as a onetime key, was patented in 1918 and constitutes the basis for most cryptosystems in use today where security is of utmost concern. The basic concept of the Vernam cipher is simple, using an exclusive OR operation between the plaintext and a random binary key. Since the exclusive OR is its own inverse, a similar operation will recover the plaintext at the receiving end. The security of the system lies in the fact that the key is used only once and discarded. This feature defeats the basic concept of cryptanalysis, that under heavy traffic conditions the key can eventually be determined from patterns existing in the text itself. The disadvantage of the Vernam cipher is the problem of producing, registering, distributing and cancelling the keys. Nevertheless, for systems where security is required at virtually any cost, the Vernam cipher has in the past been the only choice.

Recently, a new approach to cryptography has evolved in which the objective is not to produce a theoretically unbreakable cipher, but rather to produce a cipher which would require an inordinate effort for cryptanalysis (reference 2). For example, a cipher which would require years on the fastest computer for effective cryptanalysis is for all practical purposes unbreakable. The basis of such ciphers lies in a rapidly developing area of mathematics known as complexity theory. In addition to security, this new approach produces ciphers with a revolutionary and very useful characteristic.

All previous ciphers, including the Vernam cipher, fall into one of two general categories:

1. Public algorithm - secret key cipher, where the algorithm is public knowledge but the key is known only to the sender and receiver. The National Bureau of Standard's DES is of this type.

2. Secret algorithm - secret key cipher, where both the algorithm and the key are known only to the sender and receiver.

It is generally conceded that public algorithm systems are the more trustworthy, since one can choose an algorithm which has withstood the scrutiny of cryptanalytic experts, thus avoiding the possibility of a flaw in the algorithm which might be exploited by a penetrator. Now from the new approach comes yet another category:

3. Public algorithm - public key cipher, where both the algorithm and the key are public knowledge.

While such a concept may seem unworkable in that a person who encrypts a plaintext with a given key should also be able to decrypt the resulting ciphertext, such is not the case. By using as the encrypting algorithm a mathematical procedure known as a "one-way trapdoor function," the encryption process is quite easy using the encrypting key, but decryption is extremely difficult unless one also has knowledge of a second key known as the decrypting key. Thus, the algorithm and the encrypting key can be made public so that anyone can encrypt the plaintext, but only those possessing the decrypting key can decrypt the ciphertext. The efficiency of any

one-way trapdoor function in terms of cryptanalytic difficulty is determined by the principles of complexity theory.

Several one-way trapdoor functions have been suggested. The one considered here is the RSA public key system, named after its developers R. L. Rivest, A. Shamir, and L. Adelman. The system is based on the difficulty of factoring a very large nearly-prime number, a problem with a long and distinguished history of resisting solution. The concept of factoring a large number also illustrates the trapdoor characteristic, in that factoring a number is a much more complex task than producing the number from its factors. The basis of the RSA algorithm rests in some rather interesting characteristics of modular arithmetic which result from basic number theory. The first on these is given below:

$$(A \times B) \bmod N = ((A \bmod N) \times (B \bmod N)) \bmod N \quad (1)$$

For example, for A = 30, B = 20, and N = 7,

$$(30 \times 20) \bmod 7 = 600 \bmod 7 = 5$$
$$A \bmod N = 30 \bmod 7 = 2$$
$$B \bmod N = 20 \bmod 7 = 6$$
$$(6 \times 2) \bmod 7 = 12 \bmod 7 = 5$$

The second characteristic concerns exponential operations in modular arithmetic:

$$(A^B) \bmod N = (A^{B \bmod \phi(N)}) \bmod N \quad (2)$$

where $\phi(N)$ is known as Euler's totient function and is defined as the number of integers between 1 and N which have no common factors with N. For example, for A = 2, B = 18, N = 15,

$$(2^{18}) \bmod 15 = 262{,}144 \bmod 15 = 4$$

For N = 15, there are 8 integers between 1 and 15 which have no common factors with 15. These are

1, 2, 4, 7, 8, 11, 13, 14

Thus, $\phi(15) = 8$, and

$$2^{18} \bmod 8 = 2^2 = 4$$

and the identity of (2) is exemplified.

The RSA trapdoor function algorithm utilizes the fact that if N is the product of two prime numbers p and q,

$$N = pq \quad (3)$$

then the totient function is

$$\phi(N) = (p-1)(q-1) \quad (4)$$

This fact is also illustrated by the example above, since

$$(3)(5) = 15$$

and both 3 and 5 are prime. This gives

$$(p-1)(q-1) = (2)(4) = 8$$

which is the number of integers between 1 and 15 with no common factor other than 1 with 15. Furthermore, it is evident from (2) that, for any integer E between ϕ and N-1,

$$E^{\phi(N)+1} = E \bmod N \quad (5)$$

In addition, if E is restricted to lie in range from 3 to $\phi(N)-1$, with no common factor with $\phi(N)$, then it has a modulo $\phi(N)$ multiplicative inverse D such that

$$(ED) \bmod \phi(N) = 1 \quad (6)$$

In the RSA algorithm E is the encryption key, D is the decryption key, and expression (6) is the basic relationship of the algorithm. The power of the algorithm rests in the fact that $\phi(N)$ is easy to determine if p and q are known, but computing $\phi(N)$ directly from N is equivalent in difficulty to factoring N.

The application of the algorithm then consists of the following procedures. Select two large prime numbers p and q. Then, in accordance with the previous discussion,

$$pq = N$$
$$(p-1)(q-1) = \phi$$

Choose a random number E between 3 and ϕ which has no common factor with ϕ. The inverse D is found from (6), using an extended version of Euild's algorithm for determining the greatest common divisor of two integers. To encrypt a plaintext message P, in the form of an integer between 0 and N-1, the encrypting equation is

$$C = P^E \bmod N \quad (7)$$

where C is the resulting ciphertext. The decrypting equation is

$$P = C^{D} \bmod N \qquad (8)$$

Thus, a person who knows N and E can encrypt a plaintext message, but cannot decrypt a message encrypted by another person using the same values of N and E unless he also knows the decrypting key D. The cryptanalyst is faced with the task of factoring N, an inordinate effort for N on the order of hundreds of digits.

It is easy to see that expression (8) will recover the plaintext, since from (8), (7), (6) and (2),

$$C^{D} \bmod N = (P^{E} \bmod N)^{D} \bmod N$$

$$= P^{ED} \bmod N$$

$$= (P^{ED \bmod \phi(N)}) \bmod N$$

$$= P^{1} \bmod N = P$$

The characteristic which makes this possible is that given in expression (6). Similar development will show the symmetry of the relationship; that is, either P or C can be considered data to be encrypted.

As an example, consider the following illustrative example taken from reference 2:

$p = 73 \qquad q = 151 \qquad N = 11{,}023$

$\phi(11{,}023) = 10{,}800 \qquad E = 11$

Then, the decryption key D is determined as that integer for which

$$(D \times 11) \bmod 10{,}800 = 1$$

The value of D is determined to be

$$D = 5{,}891$$

The plaintext to be encrypted is the integer 3314. Using expression (7) results in the ciphertext:

$$C = 3314^{11} \bmod 11{,}023$$

$$= 10{,}260$$

To decrypt this ciphertext, expression (8) is used:

$$P = 10{,}260^{5{,}891} \bmod 11{,}023$$

$$= 3314$$

In this illustrative example the value of N is not large enough to frustrate cryptanalysis, since one hundred or more digits are required for absolute security. And yet, even with this simple example, exponents on the order of 6000 are encountered, an impracticality to say the least for conventional computational techniques. However, by using the property of modular arithmetic described in expression (1) and a binary representation of the encrypting or decrypting key, it is possible to evaluate expressions such as these using practical computer word lengths and in reasonable time. To illustrate this procedure, consider the encryption above, with E = 11. The binary representation of 11 is 1011 which, from the positional notation used in the binary system, actually means

$$1 \times 2^{3} + 0 \times 2^{2} + 1 \times 2^{1} + 1$$

Then,

$$C = 3314^{11} \bmod 11{,}023$$

$$= (3314^{(2^{3}+2+1)}) \bmod 11{,}023$$

$$= (3314^{8})(3314^{2})(3314) \bmod 11{,}023$$

But, from expression (1) this is equivalent to

$$C = (3314^{8} \bmod 11{,}023)(3314^{2} \bmod 11{,}023) \times (3314 \bmod 11{,}023) \bmod 11{,}023$$

This can be evaluated by noting that

$$3314^{4} \bmod 11{,}023 = (3314^{2} \bmod 11{,}023) \times (3314^{2} \bmod 11{,}023) \bmod 11{,}023$$

$$= (3688)(3688) \bmod 11{,}023$$

$$= 9985$$

In a similar fashion,

$$(3314^{8} \bmod 11{,}023) = (3314^{4} \bmod 11{,}023) \times (3314^{4} \bmod 11{,}023) \bmod 11{,}023$$

$$= (9985)(9985) \bmod 11{,}023$$

$$= 8213$$

In this manner, each of the power of two

exponentials can be evaluated modulo 11,023 from the modular value of the previous exponential. The value of C is obtained by taking the products indicated by the binary expansion of E. In this case,

$$C = (8213)(3688)(3314) \bmod 11{,}023$$

Again, the modular arithmetic can be used to formulate this as:

$$C = (8213)((3688)(3314) \bmod 11{,}023) \bmod 11{,}023$$

$$= (8213)(8545) \bmod 11{,}023$$

$$= 10{,}260$$

In this manner, the value of C is obtained from arithmetic involving no more than twice the number of digits in N. For practical systems, this is well within the capabilities of modern digital equipment.

In general terms, the above procedure consists of computing the sequence of recursive terms;

$$P^2 \bmod N$$

$$P^4 \bmod N = (P^2 \bmod N)(P^2 \bmod N) \bmod N$$

$$P^8 \bmod N = (P^4 \bmod N)(P^4 \bmod N) \bmod N$$

$$P^{16} \bmod N = (P^8 \bmod N)(P^8 \bmod N) \bmod N$$

$$P^K \bmod N = (P^{K/2} \bmod N)\ (P^{K/2} \bmod N) \bmod N$$

Where P^K is the largest power of two contained in encryption key E. None of these terms is greater than N. Then, the product of selected terms modulo N is taken, depending upon where the 1's occur in the binary representation of E. By keeping a running product modulo N, no integers greater in length than twice the length of N are encountered. The decryption process follows the same procedures, except that the binary representation of the decryption key D is used.

A useful fallout of this kind of trapdoor function encryption is the concept of the digital signature. In processes such as electronic funds transfer, it is imperative to verify the legitimacy of a digital message in terms of who really sent it. Using a trapdoor function encryption such as that described here, this is easily done. Suppose that A wishes to send a message to B, and in doing so must indicate to B that the message actually originated with A. To do this, he first decrypts the message using his secret decryption key D_A. He then encrypts this intermediate result using B's public encryption key E_B and transmits the message to B. Upon receipt of the message, B decrypts it using his own secret decrypting key D_B and obtains A's intermediate result, which was obtained by A by transforming with his secret decrypting key. Then B applies A's public encrypting key E_A to this intermediate result to obtain the original message. This is possible since the E and D transformations are inverses of one another. In this process, the data transmission is secure, since it was sent using B's encrypting key and therefore only those with B's secret decrypting key can successfully obtain the intermediate result. Furthermore, since the original plaintext is obtained from the intermediate result by applying A's public encryption key, the intermediate result must have been produced by someone possessing A's secret decryption key. Thus, B can feel secure in ascertaining that the message actually came from A.

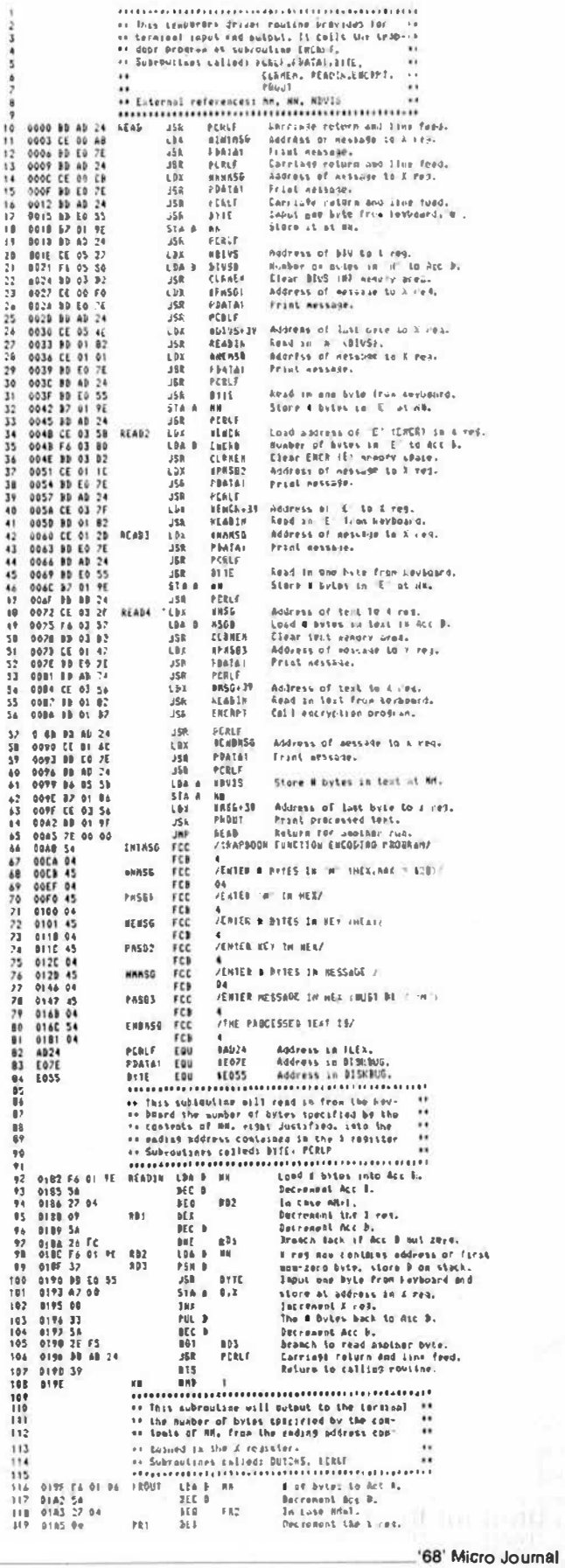
To implement this trapdoor function encryption on a general purpose computer, multi-percision multiply and divide routines are required, together with an encryption program to implement the process described above. The program described in the listing at the end of this article consists of a temporary driver routine which provides for input and output, the encryption program ENCRPT, the multiprecision multiply routine MULT, and the multiprecision divide routine DIV. The driver routine is of minimal complexity since it is intended only for experimenting with the encryption process. Further application would require a driver routine tailored to the users need. The driver routine uses two DISKBUG (or MIKBUG) subroutines, and one FLEX subroutine, PCRLF, as indicated in the listing. For the users without FLEX, PCRLF can be replaced by use of the ASCII carriage return and line feed symbols.

The multiply and divide routines are capable of handling multiprecision integers up to 40 bytes (97 decimal digits) and can be easily modified to accomodate larger integers. The routines have been kept very general, at the expense of additional complexity, in that they can efficiently handle integers with a smaller number of bytes. This is accomplished by determining the number of non-zero bytes in each integer within each routine, thus performing the basic shift-and-add and shift-and-subtract loops only as many times as necessary for the particular integers used in multiplication and division. Also, to make these routines generally useful with no external references except

subroutines, memory allocation is redundant in some cases. In the division routine, the quotient and remainder are determined, even though only the remainder is of interest in the modular division process. These features are included to make these two routines generally useful for other applications. If a user decides on the block length for his particular application, the complexity of these routines and the memory allocation can be appreciably reduced by tailoring the programs to that block length and removing the redundancy in some of the variables. As it now stands, the requirement is less than 1800 bytes, including the driver routine.

Once loaded and executed, the program will first ask for the number of bytes in N, which must then be entered in hexadecimal with leading zero's within a byte included. It will then ask for N in hexadecimal, and again a leading zero, if present, must be entered. The same process is then repeated for the key and for the message, all in hexadecimal. When this data is entered, a call to the encryption routine is made. The processed text is returned to the memory location originally containing the message and printed out. The specification of the number of bytes and the use of hexadecimal is a property of the temporary driver routine. The user's custom driver could omit the number of bytes specification and use decimal input. The number of bytes is not transferred to the computational routines; they make this determination internally, a feature incorporated to make the computational routines independent of the driver. The requirements of a driver are to place the message in memory location MSG, the value of N in memory location DIVS, and the value of the key in memory location ENCR. All of these must be right justified. In addition, it must print out the processed text from memory location MSG, where it will also be right justified. The number of non-zero bytes in the processed text is stored in NDVIS by the program.

Following the listing are shown some example input-output data, beginning with the illustrative example contained in the text.



```
                ********************************************************
                ** This tenbugers driver routine provides for    **
                ** terminal input and output. It calls the progr **
                ** depr program as subroutine ENCRPT.            **
                ** Subroutines called: PCRLF,PDATA1,BYTE,        **
                **                     CLRMEM, READIN,ENCRPT,   **
                **                     PROUT                    **
                ** External references: NN, MM, NDVIS           **
                ********************************************************
0000 BD AD 24   READ    JSR     PCRLF       Carriage return and line feed.
0003 CE 00 AB           LDX     #INMSG      Address of message to X reg.
0006 BD E0 7E           JSR     PDATA1      Print message.
0009 BD AD 24           JSR     PCRLF       Carriage return and line feed.
000C CE 00 EB           LDX     #MMMSG      Address of message to X reg.
000F BD E0 7E           JSR     PDATA1      Print message.
0012 BD AD 24           JSR     PCRLF       Carriage return and line feed.
0015 BD E0 55           JSR     BYTE        Input one byte from keyboard, &
0018 B7 01 9E           STA A   NN          Store it at NN.
001B BD AD 24           JSR     PCRLF
001E CE 05 27           LDX     #DIVS       Address of DIV to X reg.
0021 F6 05 56           LDA B   DIVSB       Number of bytes in 'N' to Acc B.
0024 BD 03 D2           JSR     CLRMEM      Clear DIVS (N) memory area.
0027 CE 00 F0           LDX     #PMSG1      Address of message to X reg.
002A BD E0 7E           JSR     PDATA1      Print message.
002D BD AD 24           JSR     PCRLF
0030 CE 05 4E           LDX     #DIVS+39    Address of last byte to X reg.
0033 BD 01 B2           JSR     READIN      Read in 'N' (DIVS).
0036 CE 01 01           LDX     #MEMSG      Address of message to X reg.
0039 BD E0 7E           JSR     PDATA1      Print message.
003C BD AD 24           JSR     PCRLF
003F BD E0 55           JSR     BYTE        Read in one byte from keyboard.
0042 B7 01 9E           STA A   NN          Store # bytes in 'E' at NN.
0045 BD AD 24           JSR     PCRLF
0048 CE 03 5B   READ2   LDX     #ENCR       Load address of 'E' (ENCR) in X reg.
004B F6 03 B0           LDA B   ENCRB       Number of bytes in 'E' to Acc B.
004E BD 03 D2           JSR     CLRMEM      Clear ENCR ('E') memory space.
0051 CE 01 1E           LDX     #PMSG2      Address of message to X reg.
0054 BD E0 7E           JSR     PDATA1      Print message.
0057 BD AD 24           JSR     PCRLF
005A CE 03 7F           LDX     #ENCR+39    Address of 'E' to X reg.
005D BD 01 B2           JSR     READIN      Read in 'E' from keyboard.
0060 CE 01 2B   READ3   LDX     #MMMSG      Address of message to X reg.
0063 BD E0 7E           JSR     PDATA1      Print message.
0066 BD AD 24           JSR     PCRLF
0069 BD E0 55           JSR     BYTE        Read in one byte from keyboard.
006C B7 01 9E           STA A   NN          Store N bytes in 'E' at NN.
006F BD AD 24           JSR     PCRLF
0072 CE 03 2F   READ4   LDX     #MSG        Address of text to X reg.
0075 F6 03 57           LDA B   #SGB        Load # bytes in text in Acc B.
0078 BD 03 D2           JSR     CLRMEM      Clear text memory area.
007B CE 01 47           LDX     #PMSG3      Address of message to X reg.
007E BD E0 7E           JSR     PDATA1      Print message.
0081 BD AD 24           JSR     PCRLF
0084 CE 03 56           LDX     #MSG+39     Address of text to X reg.
0087 BD 01 B2           JSR     READIN      Read in text from keyboard.
008A BD 01 B7           JSR     ENCRPT      Call encryption program.
                        JSR     PCRLF
008D CE 01 6C           LDX     #CHDMSG     Address of message to X reg.
0090 BD E0 7E           JSR     PDATA1      Print message.
0093 BD AD 24           JSR     PCRLF
0096 B6 05 5B           LDA A   NDVIS       Store N bytes in text at MM.
0099 B7 01 86           STA A   NN
009C CE 03 56           LDX     #MSG+39     Address of last byte to X reg.
009F BD 01 9F           JSR     PROUT       Print processed text.
00A2 7E 00 00           JMP     READ        Return for another run.
00A5 54         INTMSG  FCC     /INTAPOOON FUNCTION ENCODING PROGRAM/
00A6 04                 FCB     4
00A7 45         #MMSG   FCC     /ENTER # BYTES IN 'M' (HEX,MAX = $20)/
00C7 04                 FCB     04
00C8 45         PMSG1   FCC     /ENTER 'N' IN HEX/
00D8 04                 FCB     4
00D9 45         MEMSG   FCC     /ENTER # BYTES IN KEY (HEX)/
00F9 04                 FCB     4
00FA 45         PMSG2   FCC     /ENTER KEY IN HEX/
010A 04                 FCB     4
010B 45         MMMSG   FCC     /ENTER # BYTES IN MESSAGE /
012B 04                 FCB     04
012C 45         PMSG3   FCC     /ENTER MESSAGE IN HEX (MUST BE < 'M')
013C 04                 FCB     4
013D 54         CHDMSG  FCC     /THE PROCESSED TEXT IS/
014D 04                 FCB     4
                PCRLF   EQU     $AD24       Address in FLEX.
                PDATA1  EQU     $E07E       Address in DISKBUG.
                BYTE    EQU     $E055       Address in DISKBUG.
                ********************************************************
                ** This subroutine will read in from the key-    **
                ** board the number of bytes specified by the    **
                ** contents of NN, right justified, into the     **
                ** ending address contained in the X register    **
                ** Subroutines called: BYTE, PCRLF               **
                ********************************************************
0150 B6 01 9E   READIN  LDA B   NN          Load # bytes into Acc B.
0153 5A                 DEC B               Decrement Acc B.
0154 27 04              BEQ     RD2         In case #=1.
0156 09         RD1     DEX                 Decrement the X reg.
0157 5A                 DEC B               Decrement Acc B.
0158 26 FC              BNE     RD1         Branch back if Acc B not zero.
015A F6 01 9E   RD2     LDA B   NN          B reg now contains address of first
015D 37                 PSH B               non-zero byte, store B on stack.
015E BD E0 55   RD3     JSR     BYTE        Input one byte from keyboard and
0161 A7 00              STA A   0,X         store at address in X reg.
0163 08                 INX                 Increment X reg.
0164 33                 PUL B               The # bytes back to Acc B.
0165 5A                 DEC B               Decrement Acc B.
0166 2E F5              BGT     RD3         Branch to read another byte.
0168 BD AD 24           JSR     PCRLF       Carriage return and line feed.
016B 39                 RTS                 Return to calling routine.
016C            NN      RMB     1
                ********************************************************
                ** This subroutine will output to the terminal   **
                ** the number of bytes specified by the con-     **
                ** tents of NN, from the ending address con-     **
                ** tained in the X register.                     **
                ** Subroutines called: DUTCHS, PCRLF             **
                ********************************************************
016D F6 01 9E   PROUT   LDA B   NN          # of bytes to Acc B.
0170 5A                 DEC B               Decrement Acc B.
0171 27 04              BEQ     PR2         In case NN=1.
0173 09         PR1     DEX                 Decrement the X reg.
```


REFERENCES

Lacour, S. J. and Elbert, T. F. "A Software Data Encryption Standard Implementation for the 6800."

Hellman, M. E. "The Mathematics of Public-Key Cryptography," Scientific American, September, 1979.

```
0120 01A6 5A            DEC B          Decrement Acc B.
0121 01A7 2A FC         BNE   PR1      Branch back if Acc B not zero.
0122 01A9 F6 03 B6 PR2  LDA B AA       X reg now contains address of first
0123 01AE BD E0 CA PR3  JSR   OUT2HS   non-zero byte. Load B bytes into
0124 01AF 5A            DEC B          Acc B and output one byte. Decre-
0125 01B0 2E FA         BGT   PR3      ment Acc B and branch back for the
0126 01B2 BD AD 24      JSR   PCRLF    next byte.
0127 01B5 39            RTS            Return to calling routine.
0128 01B6               AA    RMB  1
0129 E0CA               OUT2HS EQU   $E0CA     Address in disabus.
0130                    ********************************************
0131                    ** This is the actual encryption routine which **
0132                    ** encrypts or decrypts the message. It calls  **
0133                    ** the multi-byte inlaser multiplication rou-  **
0134                    ** tine MULT and the multi-byte modular divi-  **
0135                    ** sion routine DIVSN in order to implement the **
0136                    ** functions:                                   **
0137                    **          C(I)=(P(I)+K(I))MOD(M)   encrypt    **
0138                    **          P(I)=(C(I)+K(I))MOD(M)   decrypt    **
0139                    ** where P(I) is the plaintext message and C(I) **
0140                    ** is the ciphertext message.                   **
0141                    ** Subroutines called: FIND, MOVE, CLRMEM.      **
0142                    **                     MULT, DIVSN              **
0143                    ** External references: CTR, HOLDX1, HOLDX2     **
0144                    **                     MPROD, MDIVD             **
0145                    ********************************************
0146 0137 F6 03 B0 ENCRPT LDA B ENCRB  Move B bytes in 'E' to Acc B.
0147 013A CC 03 5B      LDA   #ENCR    Address of 'E' to X reg.
0148 013D BD 03 B9      JSR   FIND     Find first non-zero byte.
0149 0140 F7 03 AC      STA B MENCR    Store B of non-zero bytes in MENCR.
0150 0143 FF 03 AE      STX   ENCRX    Store address at ENCRX.
0151 0146 F6 03 57      LDA B #56      Move B bytes in text to Acc B.
0152 0149 CC 03 2F      LDX   #MSG     Address to B req.
0153 014C BD 03 B9      JSR   FIND     Find first non-zero byte.
0154 014F F7 03 AD      STA B MMSG     Store B non-zero bytes at MMSG.
0155 0152 CE 03 82      LDX   #FINMULT the receiving area for FINMULT is
0156 0155 F6 03 57      LDA B #56      cleared except for a one in the last
0157 0158 BD 03 B2      JSR   CLRMEM   byte.
0158 015B 86 01         LDA A #01      Load Acc A with a non-zero
0159 015D B7 03 A9      STA A FINMULT+39 bytes in ENCR. The EXCE loop multa-
0160 0160 B6 03 AC      LDA A MENCR    plies this number by B to yield the
0161 0163 7F 03 AA      CLR   CTR      number of bits in ENCR. This is
0162 0166 7F 03 AB      CLR   CTR+1    stored at CTR
0163 0169 F6 03 AB EXTA LDA B CTR+1
0164 016C CB 08         ADD B #08
0165 016E F7 03 AB      STA B CTR+1
0166 0171 F6 03 AA      LDA B CTR
0167 0174 C9 00         ADC B #00
0168 0176 F7 03 AA      STA B CTR
0169 0179 4A            DEC A
0170 017A 26 ED         BNE   EXTA
0171 017C FE 03 A6      LDX   CTR      Decrement CTR and perform initial
0172 017F 09            DEX            computation of FINMULT.
0173 0180 FF 03 AA      STX   CTR
0174 0183 FE 03 AE      LDX   ENCRX    Load beginning address of ENCR into
0175 0186 0C            CLC            X reg and B bytes into Acc A. Clear
0176 0187 B6 03 AC      LDA A MENCR    save area SAVCAR. Rotate right ENCR
0177 018A 7F 03 B1      CLR   SAVCAR   and if carry is not set branch to
0178 018D 46 00      INIT ROR   0,X    ENCR1 for for subsequent computa-
0179 018F DB            IMP            tion.
0180 0210 4A            DEC A
0181 0211 26 FA         BNE   INIT
0182 0213 24 15         BCC   ENCR1
0183 0215 F6 03 AD      LDA B #A50     If carry set, make FINMULT + MSG
0184 0218 F7 03 CD      STA B CTR      then go to ENCR1 for subsequent
0185 021B CE 03 A9      LDX   #FINMULT+39 computation.
0186 021E FF 03 D0      STX   HOLDX2
0187 0221 CE 03 56      LDX   #MSG+39
0188 0224 FF 03 CE      STX   HOLDX1
0189 0227 BD 03 B0      JSR   MOVE
0190 022A FC 03 AA ENCR4 LDX  CTR      Decrement CTR. If zero branch to
0191 022D 09            DEX            end, otherwise continue.
0192 022E FF 03 AA      STX   CTR
0193 0231 2C 03         BGE   CONT
0194 0233 7E 03 49      JMP   ENCR5
0195 0236 FE 03 AE CONT LDX   ENCRX    Address of ENCR to X reg.
0196 0239 0C            CLC            Clear carry bit.
0197 023A B6 03 AC      LDA A MENCR    The B non-zero bytes in ENCR to Acc A.
0198 023D 7F 03 B1      CLR   SAVCAR   Clear the carry save area.
0199 0240 46 00      ENCR2 ROR  0,X    The multibyte contents of ENCR
0200 0242 08            INX            are rotated right one bit.
0201 0243 4A            DEC A
0202 0244 26 FA         BNE   ENCR2
0203 0246 2C 03         BCC   ENCR3    If carry not set, continue.
0204 0248 7C 03 B1      INC   SAVCAR   If set, place one in SAVCAR, continue.
0205 024B F6 03 57 ENCR3 LDA B #56      Store B bytes in MSG into CTX
0206 024E F7 03 CD      STA B CTX      and address of MULTB into HOLDX2.
0207 0251 CE 06 45      LDX   #MULTB+19 Store address of MSG into HOLDX1
0208 0254 FF 03 D0      STX   HOLDX2   and call MOVE to transfer MSG
0209 0257 CE 03 56      LDX   #MSG+39  to MULTB in preparation for
0210 025A FF 03 CE      STX   HOLDX1   call to multiplication routine.
0211 025D BD 03 B0      JSR   MOVE
0212 0260 F6 03 57      LDA B #56      Store B bytes in MSG into CTX and
0213 0263 F7 03 CD      STA B CTX      address of MULTB into HOLDX2.
0214 0266 CE 06 0D      LDX   #MULTB+3e Store address of MSG into HOLDX1
0215 0269 FF 03 D0      STX   #HOLDX2  and call MOVE to transfer MSG to
0216 026C CE 03 56      LDX   #MSG+39  location MULTR in preparation for
0217 026F FF 03 CE      STX   HOLDX1   call to multiplication routine.
0218 0272 BD 03 B0      JSR   MOVE     Jump to multiplication routine.
0219 0275 BD 05 60      JSR   MULP     Clear divide memory space.
0220 0278 CE 04 07      LDX   #DIVD
0221 027B F6 05 4F      LDA B DIVDB
0222 027E BD 03 D2      JSR   CLRMEM
0223 0281 F6 05 4F      LDA B DIVDB    Move PROD from previous multiplica-
0224 0284 F7 03 CD      STA B CTX      tion call into DIVD. The prime

0225 0287 CE 05 26      LDX   #DIVD+29 number used in the encryption al-
0226 028A FF 03 D0      STX   HOLDX2   gorithm is already in DIVD.
0227 028D CE 06 BF      LDX   #PROD+29
0228 0290 FF 03 CE      STX   HOLDX1
0229 0293 BD 03 B0      JSR   MOVE
0230 0296 BD 03 E4      JSR   DIVSN    Jump to division subroutine.
0231 0299 F6 05 50      LDA B #DVIS    Load B of non-zero bytes into
0232 029C F7 03 CB      STA B CTX      CTX and ending address of
0233 029F CE 03 56      LDX   #MSG+39  MSG into HOLDX2. Store TEMPX into
0234 02A2 FF 03 D0      STX   HOLDX2   HOLDX1 and call MOVE to transfer re-
0235 02A5 FE 05 53      LDX   TEMPX    mainder to MSG.
0236 02A8 FF 03 CE      STX   HOLDX1
0237 02AB BD 03 B0      JSR   MOVE
0238 02AE B6 03 B1      LDA A SAVCAR   If SAVCAR byte was turned on pre-

0239 02B1 27 63         BEQ   ENCR4    viously then fall through,else branch
0240 02B3 F6 03 57      LDA B #MSG     to BBBD.
0241 02B6 F7 03 CD      STA B CTX      Multiplication is performed here in
0242 02B9 CE 06 6D      LDX   #MULTR+39 order to keep a running product when
0243 02BC FF 03 D0      STX   HOLDX2   when SAVCAR is 1.
0244 02BF CE 03 56      LDX   #MMSG+39
0245 02C2 FF 03 CE      STX   HOLDX1   Move MSB to MULTR for the multipli-
0246 02C5 BD 03 B0      JSR   MOVE     cation routine.
0247 02C8 F6 03 57      LDA B #MSG     Move FINMULT to MULTB for the multi-
0248 02CB F7 03 CD      STA B CTX      plication routine.
0249 02CE CE 06 65      LDX   #MULTB+39
0250 02D1 FF 03 D0      STX   HOLDX2
0251 02D4 CE 03 A9      LDX   #FINMULT+39
0252 02D7 FF 03 CE      STX   HOLDX1
0253 02DA BD 03 B0      JSR   MOVE
0254 02DD BD 05 60      JSR   MULT     Jump to multiplication routine.
0255 02E0 CE 04 07      LDX   #DIVD    Clear divide memory space.
0256 02E3 F6 06 C9      LDA B #PROD
0257 02E6 BD 03 D2      JSR   CLRMEM
0258 02E9 F6 05 4F      LDA B DIVDB    In order to keep the product FINMULT
0259 02EC F7 03 CD      STA B CTX      from expanding beyond the memory
0260 02EF CE 05 26      LDX   #DIVD+29 allocation it must be modularly
0261 02F2 FF 03 D0      STX   HOLDX2   divided each time.
0262 02F5 CE 06 BF      LDX   #PROD+29
0263 02F8 FF 03 CE      STX   HOLDX1   Move PROD to DIVD to set up the mod-
0264 02FB BD 03 B0      JSR   MOVE     ular division routine.
0265 02FE BD 03 14      JSR   DIVSN    Jump to the division routine.
0266 0301 F6 05 50      LDA B #DVIS    Move contents of DIVD defined by the
0267 0304 F7 03 CB      STA B CTX      address in TEMPX to FINMULT.
0268 0307 CE 03 A9      LDX   #FINMULT+3e
0269 030A FF 03 D0      STX   HOLDX2
0270 030D FE 05 53      LDX   TEMPX
0271 0310 FF 03 CE      STX   HOLDX1
0272 0313 BD 03 B0      JSR   MOVE
0273 0316 73 02 2e ENCR4 JMP  ENCR1    Return for another pass.
0274 0319 F6 05 55 ENCR5 LDA B #DIVB   Remainder is in FINMULT. It is
0275 031C F7 03 C0      STA B CTX      moved to MSG for return to calling
0276 031F CC 03 56      LDX   #MMSG+3e routine.
0277 0322 FF 03 D0      STX   HOLDX2
0278 0325 CE 03 A9      LDX   #FINMULT+39
0279 0328 FF 03 CE      STX   HOLDX1
0280 032B BD 03 B0      JSR   MOVE
0281 032E 39            RTS            Return to calling routine.
0282 032F               ASG   RMB  05
0283 0357 20            ASGB  FCB  40
0284 0358               ENCB  RMB  40
0285 0380 28            ENCBB FCB  40
0286 0381 00         .  SAVCAR FCB  0
0287 0382               FINMULT RMB  40
0288 03AA               CTR   RMB  2
0289 03AC               #ENCB RMB  1
0290 03A0               MMSG  RMB  1
0291 03AE               ENCRX RMB  2
0292                    ********************************************
0293                    ** This routine will move any number of bytes **
0294                    ** in memory specified by the ending byte     **
0295                    ** HOLDX1 to another location in memory speci- **
0296                    ** fied by the ending byte HOLDX2. The number **
0297                    ** of bytes to be moved is specified by CTX.  **
0298                    ********************************************
0299 0380 E6 00         MOVE  LDA B 0,X    Load first byte to be moved into
0300 0382 FE 03 CE      LDX   HOLDX1   Acc B, load 1 out from HOLDX1.
0301 0385 09            DEX
0302 0386 FF 03 CE      STX   HOLDX1   Decrement the index register and
0303 0389 FE 03 D0      LDX   HOLDX2   store result in HOLDX1.
0304 038C E7 00         STA B 0,X      Load HOLDX2 into the index register
0305 038E 09            DEX            and store accumulator B into the
0306 038F FF 03 D0      STX   HOLDX2   receiving byte.
0307 0392 7A 03 CD      DEC   CTX     Decrement the index register and
0308 0395 27 05         BEQ   MOVE1    store in HOLDX2.
0309 0397 FE 03 CE      LDX   #DEDX1   Decrement CTX (counter), when 0
0310 039A 20 E4         BRA   MOVE     RTS,else load HOLDX1 into index reg-
0311 039C 39            MOVE1 RTS      ister and branch to MOVE for next byte.
0312 039D 00            CTR   FCB  0
0313 039E 00 00         HOLDX1 FDB  00
0314 03A0 00 00         HOLDX2 FDB  00
0315                    ********************************************
0316                    ** This routine clears memory up to 255 bytes **
0317                    ** as determined by the contents of Acc B,    **
0318                    ** starting at the address specified by the   **
0319                    ** contents of the index register.            **
0320                    ********************************************
0321 03A2 6F 00         CLRMEM CLR  0,X    Clears contents specified by index
0322 03A4 08            INX            register. Increment index register.
0323 03A5 5A            DEC B          Decrement counter, if not zero
0324 03A6 26 FA         BNE   CLRMEM   branch back to beginning of loop.
0325 03A8 39            RTS
0326                    ********************************************
0327                    ** This routine determines the number of non- **
0328                    ** zero bytes in a multi-byte word whose add- **
0329                    ** dress is in the X register and whose maxi- **
0330                    ** mum number of bytes is in Accumulator B. Af- **
0331                    ** ter execution Accumulator B contains the   **
0332                    ** number of non-zero bytes, and X + B register **
0333                    ** contains the address of the first non-zero **
0334                    ** byte.                                      **
0335                    ********************************************
0336 03A9 09            FIND  DEX              Decrement X reg and ACC B in preparation
0337 03AA 5C            INC B                  for loop+eq.
0338 03AB 5A            FIND1 DEC B            Decrement Acc B and check for
0339 03AC 27 05         BEQ   FIND2            zero. If zero,exit.
0340 03AE 08            INX                    Increment X reg.
0341 03AF A6 00         LDA A 0,X              Load ACC A from address in X reg.
0342 03B1 27 F8         BEQ   FIND1            If zero, branch back.
0343 03B3 39            FIND2 RTS              Return to calling routine.
0344                    ********************************************
0345                    ** Modular division routine provides for divi- **
0346                    ** sion of a multibyte (1-80 bytes) decimal num- **
0347                    ** ber (DIVD) by another multiple byte (1-40  **
0348                    ** bytes) decimal number (DIVB), giving the re- **
0349                    ** mainder in the high order bytes of DIVD,   **
0350                    ** and the quotient in the low order bytes of **
0351                    ** DIVD. The remainder will be right justified **
0352                    ** at the address contained in TEMPX, and the **
0353                    ** quotient in the remaining lower significant **
0354                    ** bytes,right justified at DIVD+29.          **
0355                    ** Subroutines used: FIND                     **
0356                    ** External references: None                  **
0357                    ********************************************
```

```
03B0 03E4 F6 05 50  DIVBN   LDA B   DIVBB     Load max B bytes into Acc B.
03B3 03E7 CE 05 27          LDX     #DIVS     Load address of DIVS into X reg.
03B6 03EA BD 03 D9          JSR     FIND      Call FIND to determine non-zero bytes.
03B9 03ED F7 05 5B          STA B   NDVIB     Store B non-zero bytes at NDIVB.
03BC 03F0 FF 05 59          STX     DIVSX     Store address of first non-zero at DIVSX.
03BF 03F3 F6 05 4F          LDA B   DIVBB     Load max B bytes into Acc B.
03C2 03F6 CE 04 D7          LDX     #DIVD     Load address of DIVD into X reg.
03C5 03F9 BD D2 3e          JSR     FIND      Determine non-zero bytes.
03C8 03FC 5C                INC B             Increment and store B bytes at #DIVD.
03C9 03FD F7 05 55          STA B   NDIVD
03CC 0400 09                DEX               Decrement and store address at DIVDX.
03CD 0401 FF 05 56          STX     DIVDX     This assures extra byte in DIVD.
03D0 0404 B6 05 55          LDA A   NDIVD     Determine B non-zero bytes in re-
03D3 0407 B0 05 5B          SUB A   NDVIS     mainder, and use DCTR loop to multi-
03D6 040A 7F 05 51          CLR     CTRD      ply this number by B to give num-
03D9 040D 7F 05 52          CLR     CTRD+1    ber of bits in remainder. Store this
03DC 0410 F6 05 52  DCTR    LDA B   CTRD+1    at CTRD.
03DF 0413 CB 08             ADD B   #08
03E1 0415 F7 05 52          STA B   CTRD+1
03E4 0418 F6 05 51          LDA B   CTRD
03E7 041B C9 00             ADC B   #00
03E9 041D F7 05 51          STA B   CTRD
03EC 0420 4A                DEC A
03ED 0421 26 ED             BNE     DCTR
03EF 0423 CE 05 26          LDX     #DIVD+79  Store ending address of DIVD at
03F2 0426 FF 05 53          STX     TEMPX     TEMPX.
03F5 0429 B6 05 54          LDA A   TEMPB+1   Load LSB into Acc A and 8 of bytes
03F8 042C F6 05 53          LDA B   NDIVD     in dividend into ACC B. Subtract
03FB 042F F0 05 5B          SUB B   NDVIS     NDVIS. If NDVIS>NDIVD jump to end.
03FE 0432 22 03             BHI     CHK       Otherwise continue and subtract
0400 0434 7E 04 B6          JMP     DV9       (NDIVD-NDVIS) from NDIVD+79 to de-
0403 0437 10        CHK     SBA               termine TEMPX, the byte where shift-
0404 0438 B7 05 54          STA A   TEMPX+1   ing occurs.
0407 043B B6 05 53          LDA A   TEMPX
040A 043E 82 00             SBC A   #00
040C 0440 B7 05 53          STA A   TEMPX
040F 0443 7F 05 5B  DV1     CLR     DIVCAR    Clear carry save area.
0412 0446 FE 05 56          LDX     DIVDX     Store starting address of DIVD in
0415 0449 FF 05 5E          STX     STORX2    STORX2, starting address of DIVS in
0418 044C FE 05 59          LDX     DIVSX     STORX1, and B of bytes in Acc B.
041B 044F FF 05 5C          STX     STORX1
041E 0452 F6 05 5B          LDA B   NDVIS
0421 0455 FE 05 5E  DV2'    LDX     STORX2
0424 0458 A6 00             LDA A   0,X       Load Acc A with a byte of DIVD and
0426 045A FE 05 5C          LDX     STORX1    compare with corresponding byte in
0429 045D A1 00             CMP A   0,X       DIVS. If DIVD<DIVS branch to shift.
042B 045F 25 40             BCS     DV5
042D 0461 26 0E             BNE     DV3       If DIVD=DIVS increment and return
042F 0462 08                INX               to compare. If DIVD>DIVS branch to
0430 0464 FF 05 5C          STX     STORX1    DV3 to accomplish subtraction.
0433 0467 FE 05 5E          LDX     STORX2
0436 046A 08                INX
0437 046C FF 05 5E          STX     STORX2
043A 046F 5A                DEC B
043B 0470 26 E4             BNE     DV2
043D 0471 CE 05 4E  DV3     LDX     #DIVD+39  Store ending address of DIVS into
0440 0474 FF 05 5C          STX     STORX1    STORX1. Set carry save DIVCA to one.
0443 0477 7C 05 5B          INC     DIVCAR
0446 047A FE 05 53          LDX     TEMPX     Store TEMPX into STORX2, clear carry,
0449 047D FF 05 5E          STX     STORX2    and store B of bytes into Acc B.
044C 0480 0C                CLC
044D 0481 F6 05 5B  DV4     LDA B   NDVIB
0450 0484 A6 00     DV4     LDA A   0,X       Enter DV4 loop to subtract bytes in
0452 0486 FE 05 5C          LDX     STORX1    STORX1 from corresponding bytes in DIVD.
0455 0489 A2 00             SBC A   0,X
0457 048B FE 05 5E          LDX     STORX2
045A 048E A7 00             STA A   0,X       Store results in DIVD.
045C 0490 09                DEX
045D 0491 FF 05 5E          STX     STORX2
0460 0494 FE 05 5C          LDX     STORX1
0463 0497 09                DEX
0464 0498 FF 05 5C          STX     STORX1
0467 049B FE 05 5E          LDX     STORX2
046A 049E 5A                DEC B             When all bytes have been subtracted
046B 049F 26 E3             BNE     DV4       continue to shift.
046D 04A1 FE 05 51  DV5     LDX     CTRD      Decrement bit counter. If < 0 branch
0470 04A4 09                DEX               to end.
0471 04A5 FF 05 51          STX     CTRD
0474 04A8 2B 16             BMI     DV7
0476 04AA F6 05 55          LDA B   NDIVB     Otherwise, load Acc B with B of bytes
0479 04AD CE 05 26          LDX     #DIVD+39  and clear carry.
047C 04B0 0C                CLC
047D 04B1 B6 05 5B          LDA A   DIVCAB    If DIVCAR is one, set carry and shift.
0480 04B4 27 01             BEQ     DV6       Otherwise go directly to shift.
0482 04B6 0D                SEC
0483 04B7 69 00     DV6     ROL     0,X       Shift DIVD left.
0485 04B9 09                DEX
0486 04BA 5A                DEC B
0487 04BB 26 FA             BNE     DV6
0489 04BD 7E 04 43          JMP     DV1       Return for another pass.
048C 04C0 CE 05 26  DV7     LDX     #DIVD+39
048F 04C3 F6 05 55          LDA B   NDIVD     When all shifting of ENCR is complete
0492 04C6 F0 05 5B          SUB B   NDVIS     perform final shift, with carry bit
0495 04C9 0C                CLC               if DIVCAR=1, carry clear if DIVCAR=0.
0496 04CA B6 05 5B          LDA A   DIVCAB
0499 04CD 27 01             BEQ     V8
049B 04CF 0D                SEC
049C 04D0 69 00     DV8     ROL     0,X
049E 04D2 09                DEX
049F 04D3 5A                DEC B
04A0 04D4 26 FA             BNE     DV8
04A2 04D6 39        DV9     RTS               Return to calling routine.
04A3 04D7           DIVD    RMB     39
04CA 0527           DIVS    RMB     40
04CC 054F           DIVDB   FCB     68
04CD 0550           DIVBB   FCB     40
04CE 0551           CTRD    RMB     2
04D0 0553           TEMPX   FDB     00
04D2 0555           NDIVD   RMB     1
04D3 0556           DIVDX   RMB     2
04D5 0558           DIVCAR  RMB     1
04D6 0559 00 00     DIVBX   FDB     00
04D8 055B           NDVIS   RMB     1
04D9 055C           BTORX1  RMB     2
04DB 055E           BYORX2  RMB     2
                    ********************************************
                    ** Integer multiply routine Provides for multi- **
                    ** plication of two multi-byte (1-69 bytes)    **
                    ** integer numbers MULTR (multiplier) and      **
                    ** MULTD (multiplicand), giving the product in **
                    ** PROD (2-80 bytes). The multiplier, multipli- **
                    ** cand, and product are right justified.      **
                    ** Subroutines called: FIND, CLRMEM            **
                    ** External references: None                   **
                    ********************************************
04DD 0560 56 04 AF  MULT    LDA B   MULTRB    Load max B bytes and address of
04E0 0563 CE 06 46          LDX     #MULTR    MULTR into Acc B and X reg, and
04E3 0566 BD 03 D9          JSR     FIND      call FIND to determine non-zero
04E6 0569 F7 06 C3          STA B   NMULTR    bytes. Store B in NMULTR and
04E9 056C FF 06 C4          STX     MULTRX    address at MULTRX.
04EC 056F F6 06 AE          LDA B   MULTDB    Load max B bytes and address of
04EF 0572 CE 06 1E          LDX     #MULTD    MULTD into Acc B and X reg, and
04F2 0575 BD 03 D9          JSR     FIND      call FIND to determine non-zero
04F5 0578 5C                INC B             bytes. Increment to ensure space
04F6 0579 F7 06 CB          STA B   NMULTD    for overflow and store at NMULTD.
04F9 057C F0 06 C3          SUB B   NMULTR    Determine K non-zero bytes in PROD.
04FC 057F F7 06 D9          STB B   NPROD
04FF 0582 0C                CLC               Clear carry.
0500 0583 B6 06 C3          LDA A   NMULTR    Load B bytes in MULTD into Acc A.
0503 0586 7F 06 C1          CLR     CTRM      Clear CTRM.
0506 0589 7F 06 C2          CLR     CTRM+1
0509 058C F6 06 C2  MULP    LDA B   CTRM+1    Value of CTRM+1 into Acc B.
050C 058F CB 08             ADD B   #08       Increase CTRM+1 by eight.
050E 0591 F7 06 C2          STB B   CTRM+1    Load CTRM into Acc B and add with
0511 0594 F6 06 C1          LDA B   CTRM      carry to zero so to have the full
0514 0597 C9 00             ADC B   #00       sum in a two byte location at CTRM
0516 059A F7 06 C1          STA B   CTRM      and CTRM+1. This loop determines
0519 059C 4A                DEC A             8 of bits in MULTR.
051A 059D 26 ED             BNE     MULP
051C 059F CE 06 70          LDX     #PROD     Clear entire space allocation for
051F 05A2 F6 06 C0          LDA B   PRODB     PROD (the multi-byte product.)
0522 05A5 BD 03 D2          JSR     CLRMEM    Size is specified by Acc B.
0525 05A8 CE 06 9F          LDX     #PROD+79  Determine PRODX, the beginning add-
0528 05AB FF 06 CE          STX     SHFTX     ress of the product.
052B 05AE 08                INX
052C 05AF FF 06 C6          STX     PRODX
052F 05B2 B6 06 C7          LDA A   PRODX+1
0532 05B5 B0 06 C9          SUB A   NPROD
0535 05B8 B7 06 C7          STA A   PRODX+1
0538 05BB B6 06 C6          LDA A   PRODX
053B 05BE 82 00             SBC A   #00
053D 05C0 B7 06 C6          STA A   PROD1
0540 05C3 B6 06 CF          LDA A   SHFTX+1   Determine SHIFTX, the address in
0543 05C6 B0 06 C3          SUB A   NMULTR    PROD where shifting and addition
0546 05C9 B7 06 CF          STA A   SHFTX+1   occurs.
0549 05CC B6 06 CE          LDA A   SHFTX
054C 05CF 82 00             SBC A   #00
054E 05D1 B7 06 CE          STA A   SHFTX
0551 05D4 0C        MP1     CLC               Clear carry.
0552 05D5 FE 06 C4          LDX     MULTRX    LOAD index reg with value of MULTR.
0555 05D8 F6 06 C3          LDA B   NMULTR    Load B non-zero byte in-
0558 05DB 66 00     MP1A    ROR     0,X       to Acc B. Shift right the
055A 05DD 08                INX               the X reg. Each byte of MULTR
055B 05DE 5A                DEC B             is shifted in this loop.
055C 05DF 26 FA             BNE     MP1A
055E 05E1 24 24             BCC     MP3       If carry cleared from last ROR 0,X
0560 05E3 FE 06 CE          LDX     SHFTX     branch to shift. Otherwise, store
0563 05E6 FF 06 CC          STX     TEMPX2    SHIFTX in TEMPX2 and ending add-
0566 05E9 CE 06 45          LDX     #MULTR+39 ress of MULTD in X reg.
0569 05EC 0C                CLC               Clear the carry bit.
056A 05ED F6 06 C8          LDA B   NMULTB    B bytes in MULTD in ACC B.
056D 05F0 A6 00     MP2A    LDA A   0,X       The value at the address contained
056F 05F2 09                DEX               the X reg is loaded into Acc A.
0570 05F3 FF 06 CA          STX     TEMPX1    This is a byte of MULTD, X reg
0573 05F6 FE 06 CC          LDX     TEMPX2    is decremented and stored in TEMPX1.
0576 05F9 A9 00             ADC A   0,X       The address contained in TEMPX2 is
0578 05FB A7 00             STA A   0,X       byte of PROD) is loaded into X reg. reg
057A 05FD 09                DEX               The contents are added and the pro-
057B 05FE FF 06 CC          STX     TEMPX2    cess continued for each successive
057E 0601 FE 06 CA          LDX     TEMPX1    byte in MULTB.
0581 0604 5A                DEC B             When loop completed, clear carry.
0582 0605 26 E9             BNE     MP2A
0584 0607 0C        MP3     CLC
0585 0608 FE 06 C6          LDX     PRODX     Load the address of PROD into X
0588 060B F6 06 C9          LDA B   NPROD     reg and number of bytes in PROD
058B 060E 66 00     MP4A    ROR     0,X       (NPROD) into Acc B. Rotate PROD
058D 0610 08                INX               right to test running total. This
058E 0611 5A                DEC B             loop will rotate every byte in PROD.
058F 0612 26 FA             BNE     MP4A
0591 0614 FE 06 C1          LDX     CTRM      Decrement bit counter.
0594 0617 09                DEX
0595 0618 FF 06 C1          STX     CTRM
0598 061B 26 B7             BNE     MP1       Branch back for another shift.
059A 061D 39        MP6     RTS               Return to calling routine.
059B 061E           MULTD   RMB     40
059C 0646           MULTR   RMB     40
059D 066E           MULTDB  FCB     40
059E 066F           MULTR3  FCB     40
059F 0670           PROD    RMB     80
05A0 06C0           PRODB   FCB     80
05A1 06C1           CTRM    FDB     00
05A3 06C3           NMULTR  RMB     1
05A4 06C4           MULTRX  RMB     2
05A6 06C6           PRODX   RMB     2
05A7 06C8           NMULTD  RMB     1
05A8 06C9           NPROD   RMB     1
05A9 06CA           TEMPX1  RMB     2
05AB 06CC           TEMPX2  RMB     2
05AD 06CE           SHFTX   RMB     2
05AF 06D0                   END
```

NO ERROR(S) DETECTED

SYMBOL TABLE:

```
0XYZ  E055  CHK   0437  CLRMEM 02B2  CONT  0256  CTD   03AA
CTRD  0551  CTRB  04C1  CTX   02C0  DCTR  0410  DIVCAR 0558
DIVD  04D7  DIVDB 054F  DIVDX 0554  DIVS  0527  DIVSB 0550
DIVBN 03E4  DIVSX 0559  DV1   0443  DV2   0455  DV3   0471
DV4   0484  DV5   04A1  DV6   04B7  DV7   04C0  DV8   04D0
DV9   04D6  ECTD  01E9  ENCR  0350  ENCR1  022A  ENCR2  0240
ENCR3  0249  ENCR4  0316  ENCR5  0319  ENCRB  0380  ENCRPT 0187
ENCRX  038C  ENDAS6 014E  FIND  03D9  FIND1  03DB  FIN32  03E3
FINNUL 0302  HOLDX1 03CE  HOLDX2 03D0  INIT  020D  INTH80 0048
MN    0184  MOVE  03B0  MOVE1 03CC  MP1   05D4  MP1A  05DB
MP2A  05F0  MP3   0607  MP4A  060E  MP6   061D  MR1   022F
MSGB  0357  MULP  05BC  MULT  0560  MULTD 061E  MULTDB 066E
```

```
MULTB  0440   MULERR 046F   BRLTBK 04C4   MBIVB  0555   ZDVIB  0539
MCAR6  0101   MEMCR  03AC   BMBB   B12B   BRSB   03AB   MMULTB 04C8
MMULTR 04C3   ab     0191   BORBB  00CB   BOPBB  04C9   OUT2HS E0CA
PCRLF  A024   PBATA1 E07E   P0001  00F0   PMSG2  011C   PMSG3  0142
PR1    01A5   PR2    01A9   PR3    01AC   PROB   0670   PROBB  04C0
PROBR  04C6   PROV1  0199   RB1    0188   RB2    018C   RB3    0187
RERD   0000   READ2  0040   READ3  0060   READ4  0072   READIN 0182
SAVCAR 0381   SNFTX  04CE   STBBZI 055C   STORX2 055E   TEMPX  0553
TEMPX1 04C8   TEMPX2 04CC

...ASM 0000
TRAPDOOR FUNCTION ENCODING PROGRAM
ENTER # BYTES IN 'N' (HEX,MAX = 020)
02
ENTER 'N' IN HEX
200F
ENTER # BYTES IN KEY (HEX)
01
ENTER KEY IN HEX
00
ENTER # BYTES IN MESSAGE
02
ENTER MESSAGE IN HEX (MUST BE < 'N')
0CF2

THE PROCESSED TEXT IS
20 14

TRAPDOOR FUNCTION ENCODING PROGRAM
ENTER # BYTES IN 'N' (HEX,MAX = 020)
02
ENTER 'N' IN HEX
200F
ENTER # BYTES IN KEY (HEX)
02
ENTER KEY IN HEX
1A0J
ENTER # BYTES IN MESSAGE
02
ENTER MESSAGE IN HEX (MUST BE < 'N')
2014

THE PROCESSED TEXT IS
0C F2

TRAPDOOR FUNCTION ENCODING PROGRAM
ENTER # BYTES IN 'N' (HEX,MAX = 020)

...ASM 0000
TRAPDOOR FUNCTION ENCODING PROGRAM
ENTER # BYTES IN 'N' (HEX,MAX = 020)
04
ENTER 'N' IN HEX
0CFCE720
ENTER # BYTES IN KEY (HEX)
03
ENTER KEY IN HEX
304030
ENTER # BYTES IN MESSAGE
03
ENTER MESSAGE IN HEX (MUST BE < 'N')
123456

THE PROCESSED TEXT IS
01 4E 33 B0

TRAPDOOR FUNCTION ENCODING PROGRAM
ENTER # BYTES IN 'N' (HEX,MAX = 020)
04
ENTER 'N' IN HEX
0CFCE720
ENTER # BYTES IN KEY (HEX)
01
ENTER KEY IN HEX
13
ENTER # BYTES IN MESSAGE
04
ENTER MESSAGE IN HEX (MUST BE < 'N')
014E33B0

THE PROCESSED TEXT IS
6D 12 J4 56

TRAPDOOR FUNCTION ENCODING PROGRAM
ENTER # BYTES IN 'N' (HEX,MAX = 020)
```

JCP OVERVIEW REVIEW

Ever once in a while a piece of software comes along that indicates a quantum jump in the state of the art. This months review deals with such a program, JCP, a Job Control Program written by Peter Murray of P. O. Box 49302, Austin, Texas. It is sold by the Frank Hogg Dental Lab, 130 Midtown Plaza, 700 East Water Street, Syraucse, N. Y. 13210. It sells for $39.95. Most of us working with micros have gotten used to the idea that we must load and execute every program individually and take care of any errors as they occur. In other words, we have been handling our "job stream" manually. We could have used the EXEC utility which is supplied with the FLEX operating system to execute a text file which contained a list of FLEX commands. But, with EXEC we could not handle errors or parameter substitution.

History

Peter Murray was introduced to computers on an IBM 360 system at a large university and became accustomed to using a Job Stream Processor to batch process. He was spoiled by the 360 DOS which was essentially a programming language with an instruction set composed of operating system commands, user data and other special commands whic directed program flow. He had grown accusomed to "programming" a job stream and then letting the 360 DOS do all his processing while he took care of other studys. When he began using his S550 computer system he soon realized that he did not want to give up his big system habits. So, he went to work and JCP was born.

Design Goals

His design goals called for the cabability of branching, parameter substitution, error recovery, interrupt and operator prompting. All of these features are found in the release of JCP tested. Common job control features that are not provided by this version include a common system area, a timer interrupt and input/Output scheduling. Branching becomes necessary in a job stream when it comes time to change the job sequence. JCP provides both conditional and unconditional branching which allows the operator to change a sequence without physically rearranging his job streams. Parameter substitution allows the operator to use standardized routines or utility procedures for many different jobs. He simply provides real values for the dummy values placed in the canned procedure by entering them in a parameter list following the name of the procedure file in the command line. JCP also allows you to assign values to the parameters within the procedure. This allows for defaults when the particular parameter is not provided in the command line. If you are a perfect programmer and don't make simple mistakes, you won't need JCP's error recovery feature. But, they are there any way for people like me. JCP allows you to anticipate certain expected errors, trap them and provide a method of handling them. How many times have you sent a long command to your computer and then realized that you made a mistake. Wouldn't it have been nice to be able to interrupt the processing and then be able to restart it after making a correction. JCP provides this capability. Operator prompting allows the control program to communicate with the operator. It makes monitoring the system's operation a lot simpler. JCP allows the message to go to your terminal or to a printer if the FLEX "P" command is used before JCP in the calling line.

Implementation

JCP, as distributed by the Frank Hogg Dental Lab, is written to work with FLEX 2.0. It is also available for MSI FLEX 1.0, 8-inch, hard-secotred systems. For your $39.95 you receive JCP.CMD, the object code file; a complete and extremely well documented source listing on a text file; several sample procedure files; and a user's manual with many examples which show you how to use JCP's capabilities. In addition to the step by step directions in the manual you will find handy command summarys for both JCP and its resident editor. Other appendices fully explain the programs characteristics, tell you how to adapt JCP to custom systems and provide sample procedure files and text editing examples. The inclusion of an editor is one of the most interesting features of Murray's JCP since that feature is normally not a part of big system job stream processors. JCP's editor builds a procedure file much like the FLEX BUILD utility. This version however goes much further. It allows you to execute or save the file right from the editor. Essentially, if you are working on something simple you just type in the procedure, exit the insert mode and type "RUN."

Organization

JCP.CMD is a transient command that resides on your system disk when it is not being used. "JCP, FILENAME'optional parameter list" is the the format. JCP(CR) will load in the job control processor, place its editor in the insert mode and wait for you to type your procedure file. A procedure file is a series of lines separated by a carrige return. Each line is a JCP statement, an input for a calling program, a label, or a comment. Program flow is sequential, or line by line, unless JCP encounters a statement that alters program flow. These statements include: GOTO, CALL, RETURN, IF-ELSE. Two points should be made about those lines that are to be used as input for a program. First, if a colon ":" is used in the first character position of the line in the procedure file, the line will be returned without the carriage return. Obviously this is the way you want to type it when dealing with a calling program that accepts input on a character basis. And second, if you have a statement that you intend to be input for a calling program, but it contains a group of characters that could be recognized as a JCP statement, all you need to do is type a slash "/" as the first character of the line. This tells JCP not to execute that line.

Parameter Substitution

Parameter substitution is used to help generalize a procedure file. Here's an example.

```
EOFF
ONERROR GOTO ECHON
. START
COPY 1, 2
. L1
:N
GOTO L1
. ECHON
EON
ONERROR END
GOTO START
```

The procedure above is not generalized. Everytime it runs it will copy every file on drive 1 to drive 2. If a file exists it will answer the FLEX query "Delete Original?" with "N." But, what happens if you don't want to copy your disks in the same manner each time? It's easy to rewrite lines three and five of your procedure so that it looks like this:

```
EOFF
ONERROR END
COPY $1, $2
GOTO L1
```

Now, assuming that you have named your procedure file COPY.TXT, you can execute it from FLEX by typing JCP, COPY'1'0'Y. This command line will copy everything on the disk in drive 1 to the disk in drive 0. If it finds a file which already exists it will tell FLEX to go ahead and delete the original. Up to nine parameters may be passed to a procedure file. They may be up to 30 characters in length.

Comments

A comment is entered in your procedure file by typing an asterisk "*" in the first column followed by a non-alphanumeric delimiter. This allows you to both document your procedures and send prompts to the terminal.

Labels

Label lines are the targets of the JCP statements GOTO and CALL. They begin with a period in the fist column followed by a non-alphanumeric character.

JCP Statements

The following is a summary of JCP commands.

COMMAND	FUNCTION
GOTO LABELNAME	Branch to labelname
BREAK	suspend processing
BREAKN	suspend processing after next line
CONT	continue processing
IFSET COMMAND	execute COMMAND if condition code set
IFCLR COMMAND	execute COMMAND if condition code clear
SET	set the condition code
CLR	clear the condition code
$n=string	replace parameter n with string
IF $n=string	continue with the next line if condition is true, else continue at ELSE.
IFN $n=string	opposite of above
CALL LABELNAME	branch to LABELNAME and continue until encountering RETURN
RETURN	return to the line following the last CALL
ONERROR command	execute command in event of an error
CLEAR	clear the JCP error flag
END	end the procedure
&string	deliver string to FLEX for execution continue with next line in procedure file
EON	echo all lines of the procedure file
EOFF	do not echo JCP command lines--calling program lines are echoed
^(uparrow)	is output as first character of a line if the preceeding line was a processed JCP statement

A few notes might help here. BREAK is useful because it allows you to enter a manual mode of operation in the middle of a procedure and then return to JCP control when you are finished. For example, when you need to edit some source code, etc. If your procedure file contains an error in logic and you find yourself in a dead loop, don't worry. Just type control 'C. It will return you to JCP.

Another Example

To illustrate the power of the conditional branching statements consider the following.

```
IF $1=ASM
$1=42
ASMB $1.TXT,41.CMD
ELSE
IF $1=COMPIL
```

In the above, if parameter number 1 is indeed equal to ASM then parameter one will be set equal to parameter two and the file will be assembled, etc. If it is not equal the program flow will branch to the line following the ELSE and check to see if the first parameter is equal to COMPIL. JCP has several interesting characteristics that you should know about. It can be used as a FLEX command within a procedure file to chain to another procedure. However, when the chained procedure is finished control does not return to he original procedure. And finally, JCP does not use a memory end check when loading your procedure file, nor does it check for buffer overflow on lines, parameters or labels. Be careful.

Conclusion

You probably won't realize the power of JJCP until you sit down and watch it run your computer for a half-hour of more with no operator assistance. But, you'll soon find that your only limit is your own imagination. With the ability to pass parameters back and forth, say BASIC to JCP, to sort/merge, back to BASIC, etc., you start to realize that if you interface all of your systems software together using JCP as the common bond, that you will be cutting a tremendous amount of time off of your overall programming time. My biggest problem was to come up with enough confidence to turn the operation of my computer over to JCP for the first time. Once I took that first step, I sat back and stared in amazement.

FLEX USER NOTES

BY RONALD W. ANDERSON
3540 STURBRIDGE COURT
ANN ARBOR, MI 48105

Well, Month 2 in the '68' Micro Journal is upon me already. Of course, there has not been time for feedback from my first effort to have reached me at this writing. What follows is a slight departure from my normal "Newsletter" in that It is more in the line of an article that could be called an editorial. Perhaps this will help you figure out "where I am coming from" so that you can understand my biases toward language implementations that have floating point math packages.

THE PERFECT COMPILER
AN ENGINEER'S DREAM

For about three years now, I have been involved with the application of 6800 based Microprocessor systems to machines that measure and correct unbalance in rotating parts such as crankshafts, gears, flywheels, fans, motor armatures, etc. Our uses include most of the range of applications, I believe, that are presently feasible. We have designed an analog input board that can multiplex 16 analog inputs through a program controlled gain stage and a fast 12 bit A/D converter.

The inputs to our system include PIA, ACIA, Analog, AC logic signals via Motorola input modules, a terminal for debug, a Keypad, and a digital input multiplexer of our design. Outputs include PIA, ACIA, terminal, dual 6 digit LED displays of our design, various lights, and AC output modules (solid state relays). Our most recent addition to this list is a memory mapped CRT display with both Alphanumeric and Graphics capability.

We do digital filtering of unbalance signals, rectangular - polar coordinate conversions (both ways), vector calculations, volume calculations for irregular shapes including drill tips, circular sections of milling cutters, etc. These calculations require floating point arithmetic. I've writen fast, limited accuracy, limited argument range scientific functions in Assembler for our "Math Package". These include Sine, Cosine, Arctangent, Square root, and Cube root. Our Math Package, in addition contains a number of "move" routines that work with the math routines to get data into and out of the math working area.

Programming in assembler is fine, except that we end up with 80 page programs. We have looked long and hard for a compiler that would allow us to "retire" our assembler. None that we have found to date has been suitable, though all have been excellent for one use or another. Most are disqualified for lack of floating point arithmetic.

WHAT DO WE NEED?

First of all, we don't care what language we must use. We've programmed in 10 (seriously) versions of BASIC, a mini PL/M, Fortran, A/BASIC compiler, Forth, and STRUBAL+, (a structured BASIC) and Assembler. All but Forth can produce satisfactory documentation if used properly. In fact, even Forth with enough comments can be reasonably documented. Of course, most BASIC's are interpreters and thus excluded from this discussion. Each of the compilers that we have tried have some excellent features and some poor ones.

Here, then is a list of features that would comprise our "Perfect Compiler". At the end of the required features is a list of some optional features that would make a compiler useful for systems programming as well as our dedicated applications.

VARIABLE TYPES

In order to optimize programs for memory efficiency and speed, we feel that the compiler shoule allow BYTE (8 bit), INTEGER, (16 bit), REAL or FLOATING POINT (3 byte mantissa plus one byte exponent), and CHARACTER or STRING of length specified at the varaible declaration or dimension statement. We would prefer declaring the data type rather than tacking on a $ or a % to identify string or integer types. Arrays of at least 2 dimensions are necessary for each of the above data types.

FUNCTIONS

The string functions of most BASIC's would be useful for monitor and CRT driving applications, but please deliver us from MID$(A$,7,3) "words notation".

Boolean operators for both BYTE and INTEGER variables inciuding AND, OR, NOT, and XOR are necessary for control logic applications. These operators should be spelled out and not replaced with symbols in order to improve documentation for someone unfamiliar with the language.

We would settle for the standard four function math operators with the MOD (modulo) divide function as an extra. Parenthesis should be allowed to specify the order of operations. A nest level for paren's of at least 10 should be provided. Paren's should be applicable to Boolean operations as well as arithmetic. For our dedicated processor applications, Scientific functions are not required, as we would prefer to write our own in order to be able to sacrifice accuracy and argument range for speed.

BRANCHING INSTRUCTIONS

We are practical users with practical applications, and though we know how to write structured programs as with PL/M and Pascal, we have nothing against a GOTO now and then if it simplifies a program or makes it run faster.

Loops should at least be of the DO (limit) N1, N2, STEP type like Fortran, and the inlusion of DO-WHILE and DO-UNTIL structures would be desirable. We must have a way to terminate a loop prematurely on a test. The version of Forth that we have seen does not have this feature, and some test programs that we have written are much slower than necessary because of this.

We have nothing against Fortran but the relational operator as in A .GT. B conveys no more information than A > B and there are three less characters to type! Note that only three characters (> < =) are required for all of the normal conditional tests with the <> used for "not equal".

An IF-THEN-ELSE structure is preferred to the IF-THEN, of course with the ELSE optional. The IF-THEN should allow any executable statement rather than just a GOTO. This helps eliminate GOTO in the program.

RUN TIME PACKAGE

It is most important in dedicated applications where EPROMS are used to hold the program, to reduce unnecessary memory usage. The structuring of the run time package as a library of functions that are only compiled if used in the program is very important because it does this very well.

THE COMPILE OPERATION

The compilation process should be a one or two step operation, requiring only a source text file to produce the object file. It should not require a source text to Assembler source to Relocatable object to linking load with run time package. This sort of approach produces good documentation all along the way, but four disk files are produced for every program. This process takes considerable time and operator interaction with the terminal, and it discourages polishing of the program because of the time it requires.

MATH PACKAGE

The floating point arithemtic that we require would be accurate enough if we had 4 digits, but 6 would be sufficient for almost any engineering application. A good feature would be automatic conversion to floating point where mixed mode arithmetic is called for by mixing variable types.

VARIABLE ASSIGNMENT

We need direct access to I/O ports in a way that is less cumbersome than PEEK and POKE. A/BASIC and SPL/M provide a way to define an I/O address as a variable, by setting a pointer before declaring or dimensioning that variable. This allows access by assigning a value to a variable (output) or simply using it in an expression (input).

MISCELLANEOUS REQUIREMENTS

There will always be some Assembler code required for critical speed routines, and a feature that alows imbedded assembler code is important. Also, the compiler should allow use of Hexadecimal numbers. Most of the compilers mentioned above have this feature. Only STRUBAL+ allows direct assembler code to be imbedded via mnemonics. This in conjunction with a function of STRUBAL+ that allows the programmer to get the address of a variable allows very simple insertion of Assembler code. A/BASIC allows machine (hex) code lines to be inserted preceeded by the GEN statement. SPL/M allows mnemonics if they are declared as literals whose value is the appropriate op code, as JMP LIT '7EH' where H stands for Hex. Simple access to Assembler subroutines is also important. BASIC's USER is too cumbersome.

In addition to these requirements, the compiler must be reasonably memory efficient and run reasonably fast. Motorola's Fortran, to choose an example, is only about 3 times faster than their BASIC interpreter, which is about the slowest Microprocessor BASIC around. There are now available BASIC interpreters that run much faster than this Fortran. A memory overhead factor of 2 or 2.5 as compared to the same program in Assembler would be acceptable. A/BASIC is about in this category when used for the things it can do. SPL/M is a little better but has less features.

There is another efficiency to consider, that is the efficiency of the Source code. This is not quite as important as the other efficiencies and the degree of self documenting of the language. An efficiency of about 10:1 for lines of code (Assembler vs Compiler) would be desirable but less is acceptable provided documentation is good. In this area, we find Forth to be the undisputed champion, most highly symbolic, and hardest to understand when reading someone else's program. I have a pet program to find the prime numbers from 1 to the limit of integer arithmetic for the interpreters and compilers. I've managed to get it running in all of the languages and dialects we've tried. It changes from language to language down to the algorithm used. Some compilers have features implemented that allow particular ways of testing for primality that are fast. In Forth, I was able to write the prime program with two variable assignments and three lines of code. Most of the others take about 60 lines.

Which language do we prefer? We really don't care. I'm not highly excited about the Pascal bandwaggon, as long as the syntax allows some structuring of the program. I would exclude standard BASIC because it doesn't allow variable names and labels. GOSUB 135 is meaningless, but GOSUB LINEFEED is not. GOTO START is much more meaningful than GOTO 10. IF NUMBER < MAXIMUM means much more than IF N<M. STRUBAL+ allows these two features very nicely. Pascal and PL/M both are self documenting and are useful choices.

The ability to debug a program is of course also very important. The ability to produce an Assembler source listing only as required (like A/BASIC) is very helpful. Minimizing the steps and time involved in compilation as discussed above also makes for easy debugging, because it is easy to add print statements and remove them. The availability of an interpreter that will accept and execute the same program as the compiler is the "best of all worlds". This allows interactive debug plus the speed and efficiency of running compiled code after debug is complete. In fact, with this feature, the compile process can get a little more involved without causing any problems.

It goes without saying that the compile time package must be compatible with the disk operating system. Ideally it would reference terminal and disk I/O as external jumps so that a user could interface it with any disk operating system by linking it to an I/O package (program). The documentation would have to spell out the necessary information for parameter passing and registers that need to be saved in the interface program. The

interface program could contain the necessary file open and close routines with user generated prompt strings to be compatible with the operating system. STRUBAL+ has been written in this way for easy interface to various operating systems.

BONUS FEATURES

Some real bonus features not necessary for the dedicated application but desirable to allow the use of the compiler for systems programming or engineering design programs, would be disk file handling routines in the run time package, full Scientific functions package, and output formatting. These would have to be an optional run time package perhaps again through the use of the library approach.

Output formatting could be of the Fortran type or preferrably the type used in STRUBAL+ in which the format is specified in the print statement as PRINT [7,3] , RESULT, / etc. where 7,3 specifies 7 digits total and 3 after the decimal point. The output is always placed in the output field so that the decimal points align. The output specification in STRUBAL+ is in effect until another format is specified, and so need not be repeated for each print statement in many cases. In the line of output print formatting, a TAB or SPACE function is convenient, as is a POS function (present print position). These last items are not of very high priority as it is always possible to fall back on BASIC for in-house design programs. The interpreter mode is better for one time program writing anyway.

FINAL OBSERVATIONS

I am an engineer, and I realize that all endeavors of this nature are a compromise. I am asking for a "car that can win the Indy 500 on 10 gallons of gasoline" or some other such analogy. I realize that speed, small memory usage, and features are contradictory requirements. Still, I think there is room for some further progress in the development of compilers for Microprocessors. Perhaps this article will serve as encouragement for someone to advance the state of the art. I would be curious to hear from others who agree or disagree with the requirements outlined here. Perhaps a report on how many users would be interested in "the perfect compiler" in some future issue of this magazine would be further incentive for someone to work on such a compiler.

ADDITIONAL INFORMATION

Since this was written, Lucidata's Pascal has been released in the final version that has Real (floating point) variables. Last month I published the Scientific Function procedures that would meet our needs for balancing machines. Pascal meets most of the requirements outlined above. The arithmetic is 9 digits, and speed is adequate. The addressing of I/O ports must be done in a cumbersome manner with PEEK and POKE instructions rather than the easier way of assigning a variable name to the address as described above. This can be lived with. A port may be read into an integer variable and processed as though it were an integer number. It may be multiplied by a Real number for calculation purposes, so that the inconvenience is minor. The major hangup with this Pascal, is that it is a true P code generating Pascal that therefore has a runtime Interpreter. In order for us to sell programs generated by the Pascal, it is necessary to license the runtime package for resale. I have asked Lucidata for information concerning a license and am presently waiting for a reply. Lucidata has also indicated that they could leave out some of the runtime features and make the package suitable for EPROM use. The runtime package is about 6K. Our Scientific functions require about 1K more, (we don't need all those in the Procedures published last month), and the application program codes very efficiently. A vector calculation program that I had done previously produced 500 bytes of P-code. It's source listing was a scant page, most of which was input and output statements. The program in Assembler that does this, is several pages long not including the math routines required. The debug time was

essentially that required to get the program to compile without syntax errors. Once past the compiler, it ran.

Technical Systems Consultants is working on a Pascal that will compile machine code directly, and therefore will not have the license problem, but perhaps will not be as efficient in terms of runtime package or code generation. I guess I'll have to wait and see.

To be continued next month...

PATCH SWTPC DISK BASIC VER.3 TO RUN UNDER FLEX 2.0

BY RICHARD G. CAGLE
Applevalley Day School
11103 Sagepark Lane
Houston, TX, 77089
713-481-3586 (after 6 pm)

Many of us who started out with SWTPC's MF-68 5" disk system have lots of existing programs written in the supplied SWTPC DISK BASIC VER 3... This Basic is slow compared to the newer fast Basics and does not have random access file capability. It was used with a DOS that TSC now calls MINIFLEX (altho when it is booted up in my system it calls itself 'FLEX 1'- which TSC says is the 8" disk version, which is confusing).

If you are in the process of switching over to FLEX 2 and one of the fast Basics, then you know what a pain in the neck it is to keep two different DOS's active. It would be a great convenience to be able to operate with only FLEX 2 DOS, and not have to keep two different formatted disks around.

In my own case, my business programs, which required several years to develop and debug, are quite complex. It just isn't possible to convert them overnight, and the old versions must be kept running to keep the business on track. My programs are also interrelated such that I can write General Ledger transactions from the Payroll program. Carrying this concept over to the new fast Basic versions means that all of the programs and file structures must be developed before I can start using them.

As an interim step, it is possible to use SWTPC DISK BASIC VER 3 with it running under FLEX 2.0.. The FLEX 2 documentation provides some hints on how to do this. Here is what must be done:

(a) Locate in Basic all uses of MINIFLEX constants and subroutines. These are then changed to equivalent Flex 2 addresses. The Advanced Programmer's Guide for MINIFLEX and FLEX 2 contains the addresses, which in all but one instance are compatible. The one exception is the ACIA flag which is found in MINIFLEX, but not in FLEX 2, which has a file input echo flag instead. Since my system uses an ACIA I arbitrarily picked another location that would always be non-zero - the end of memory location. If you are using an MP-C on the control port, you will have to do something else. See the listing for a hint.

(b) The length of the File Control Block (FCB) in MINIFLEX is 192 bytes, and in FLEX 2 is 320 bytes. Since Basic reserves space in the area also used by variables, following the Basic source, then the routines that make the reservation must be changed or else variables will be overwritten in the FCB area causing gross snafu!

(c) Since you no longer need to keep the $7000-$7FFF area reserved for MINIFLEX, it can be freed up for use in a system with continuous 32k rem in low memory. During startup, Basic surveys how much memory it has to use by loading a $B3 into each memory location,and checking if it stays. It does this from the end of the interpreter until $7000 or until memory runs out, returning a $FF. $7000 is MINIFLEX's line buffer and if we change it to $A080, FLEX 2's line buffer, then the memory reserving routine will run into $8000 and stop, if you are not using port 0. If you are using port 0, you should check to see what effects the writing of $B3 into it will have, or you may wish to change the line buffer address in the program to $8000. I don't believe there is any other use of the line buffer, but I have not tried it.

In attempting to locate all Basic uses of MINIFLEX addresses, I used two techniques, one was to use SWTBUG's 'F' command to find all occurrences of $70 or $71, then checking to see if these were addresses or code. This is the hard way and takes too much time. It is much easier to use a disassembler and get a printed listing of Basic and examine it. I used SWTPC's dissassembler and found that it was fairly easy to sort out the strings, constants, and code.

The routines/constants used in Basic are:

System FCB (11 uses, 2 of which were offset from FCB start address), TTYSET backspace and delete,DOS scratch memory, working drive no, line buffer and line buffer pointer, escape return address, ACIA flag, WARMS, GETCHR, PUTCHR, PCRLF, GETFIL, SETEXT, ADDBX, RPTERR, FMS CLOSE, and FMS CALL.

The program SWT.CMD loads each of the FLEX2 addresses into the index register, then stores them into the proper address in Basic as the program executes. I found that this technique would be less code that using the usual appended patch technique. Note that the last part of the program is not executed, but is loaded into the proper location for the patch. This patch moves the routine that reserves FCB space (plus 6 bytes for overhead) in the variable storage area. It moves it to the end of Basic and also modifies the $014E-$014F contents which tells Basic where its end is. It was necessary to move this routine, because of one added line of code, the 'INC B', which was needed because the 320 ($140) length of the new FCB is larger than the 256 ($FF) limit of the 'B' accumulator.

Although the instructions for use are included in the listing, they are summarized as follows:
Assemble the program as SWT.CMD and put it on a disk containing SAVE.CMD in FLEX 2.

Both MINIFLEX and FLEX 2 should be in memory, you can transfer back and forth using the WARMStart entry address.

It is a good idea to have the memory as 'clean' as possible, so you should zero all addresses between $0000 and $6FFF. The ZERO and MAP utilities published in the now defunct Flex Users Group Newsletter are good for this purpose.

While in MINIFLEX, use 'GET' to load but not execute Basic.

Exit to Flex 2 and call SWT.CMD, which will execute and return to FLEX 2. Then use the SAVE.CMD to save SWTBAS.CMD from $100 to $2500 with a transfer address of $100.

Now anytime that you desire to use SWTPC Disk Basic Version 3 under FLEX 2, simply type SWTBAS, or rename it to some name that appeals to you!

To check out the old girl, be sure to zero your memory in the range of $7000-$7FFF. This will insure that you do not get random success from a half and half interpreter using some of the routines in the old DOS.

IN CASE OF DIFFICULTY

Most problems that can be experianced will be due to your having a slightly newer or older version of Basic. Mine is version 90. In any event, it is unlikely that the addresses that you want to change have wandered very far from home. Load the unaltered Basic and manually check the contents of the addresses around the ones used in the 'STX $XXXX' lines of the program, you should find the miniflex calls to $70XX or $71XX within a few bytes above or below.

The routines that reserve the FCB area do not have a call to DOS, but are in the same form as the 'PATCH' in the listing less the 'INC B' and 'RTS'.

A disk copy of this program, both .TXT file and .CMD file is available from the undersigned. Send your disk, self addressed return packaging and return postage, or equivalent cash. Please indicate if your use is business or pleasure.

Happy Computing

Richard G. Cagle
Applevalley Day School,Inc.
11103 Sagepark Ln
Houston,TX, 77089

```
                NAM     SWT.TXT
                TTL      CONVERT      BASIC V3
TO FLEX2
                OPT     PAG
                *
                *PROG CONVERTS MINIFLEX
                *SWTPC DISK BASIC VER 3
                *TO RUN IN FLEX2.......
                *
                *WRITTEN BY:
                *   RICHARD G.  CAGLE
                *   11103 SAGEPARK LANE
                *   HOUSTON,TX,78089
                *
                *          COPYRIGHT          1980,
APPLEVALLEY DAY SCHOOL,INC.
                *
                *PUBLICATION PERMISSION
                *GRANTED TO 68 MICRO
                *JOURNAL....
                *
```

```
                *INSTRUCTIONS:
                *0.  ASSEMBLE THIS PROG
                * AS 'SWT.CMO' ON DISK
                * CONTAINING 'SAVE.CMO'
                * IN FLEX 2 FORMAT...
                *1.  WITH FLEX 2 DISK
                *  IN DRO BOOT FLEX2
                *2.  EXIT VIA 'MON'--
                * REPLACE DISK WITH
                * MINIFLEX DISK W/BASIC
                *3.  BOOT MINIFLEX
                *4.  'GET BASIC.CMD'
                *5.  EXIT MF VIA 'MON'
                *6.  'M A048  AD' (RESET
                * PC TO F2 ADDRESS)
                *7.  INSERT FLEX2 DISK
                *8.  'SWT' --RUNS THIS
                * PROGRAM
                *9.'SAVE SWTBAS.CMD,
                *     100,2500,100'
                *(SAVES MODIFIED BASIC
                *     IN FLEX 2 FORMAT)
                *10.  DONE-NOW ANYTIME
                * YOU WISH TO USE SWTPC
                * BASIC VER.  3 ,
                * TYPE 'SWTBAS'.....
                *
                *
                *
                *
3000                            ORG     $3000
3000 20 01      START   BRA     START1
3002 03         VN      FCB     3

                *FILE CONTROL BLOCK ADDR
3003 CE A8 40   START1  LDX     $A840
3006 FF 00 EE           STX     $00EE
3009 FF 0E A5           STX     $0EA5
300C FF 0E B3           STX     $0EB3
300F FF 20 E9           STX     $20E9
3012 FF 21 88           STX     $218B
3015 FF 21 9F           STX     $219F
3018 FF 21 C0           STX     $21C0
301B FF 21 07           STX     $2107
                *FILE CONTROL BLOCK NAME BYTE
301E CE A8 44           LDX     $A844
3021 FF 21 E9           STX     $21E9
                *DATA IN FCB
3024 CE A8 71           LDX     $A871
3027 FF 21 97           STX     $2197
```

```
                *TTYSET BACKSPACE
302A CE AC 00           LDX     $AC00
3020 FF 04 D2           STX     $04D2
                *TTYSET DELETE
3030 CE AC 01           LDX     $AC01
3033 FF 04 C5           STX     $04C5
                *DOS SCRATCH AREA
3036 CE AC 00           LDX     $AC00
3039 FF 0C 9E           STX     $0C9E
                *WORKING DRIVE NO.
303C CE AC 0C           LDX     $AC0C
303F FF 21 BD           STX     $21BD
                *LINE BUFFER POINTER
3042 CE AC 14           LDX     $AC14
3045 FF 0E A2           STX     $0EA2
3048 FF 1F 66           STX     $1F66
304B FF 1F 82           STX     $1F82
                *LINE BUFFER
304E CE A0 80           LDX     $A080
3051 FF 00 50           STX     $0050
                *ESC RETURN ADDRESS
3054 CE AC 16           LDX     $AC16
3057 FF 0C 7C           STX     $0C7C
305A FF 0C 8A           STX     $0C8A
                *
                * THE ACIA FLAG HAS
                *NO EQUIVALENT IN FLEX2.
                * THIS PROG ASSUMES
                *THAT AN ACIA (MP-S) IS
                *USED.  IF MP-C IS  AT
                *THE CONTROL PORT THEN
                *CHANGE ADDRESS 043E
                *(WAS BEQ) TO BRA..
                *
```

```
 3050  CE  AC 2B          LDX    $AC2B    LOAD INDEX
WITH MEMORY END
 3060  FF  04  3C          STX    $043C    USE IT
INSTEAD OF ACIA FLAG
                *WARMS
 3063 CE AD 03          LDX    $A003
 3066 FF 03 2A          STX    $032A
                *GETCHR
 3069 CE AD 15          LDX    $A015
 306C FF 01 13          STX    $0113
 306F FF 04 58          STX    $0458
                *PUTCHR
 3072 CE AD 18          LDX    $A018
 3075 FF 01 10          STX    $0110
                *PCRLF
 3078 CE AD 24          LDX    $A024
 307B FF 05 2C          STX    $052C

                *GETFIL
 307E CE AD 2D          LDX    $A020
 3081 FF 0E AE          STX    $0EAE
 3084 FF 1F 6B          STX    $1F6B
                *SETEXT
 3087 CE AD 33          LDX    $A033
 308A FF 00 FC          STX    $00FC
                *ADDBX
 308D CE AD 36          LDX    $A036
 3090 FF 1F CC          STX    $1FCC
                *RPTERR
 3093 CE AD 3F          LDX    $A03F
 3096 FF 0E FF          STX    $0EFF
                *FMS CLOSE
 3099 CE B4 03          LDX    $B403
 309C FF 00 21          STX    $0021
 309F FF 00 3B          STX    $003B
 30A2 FF 1F 19          STX    $1F19
                *FMS CALL
 30A5 CE B4 06          LDX    $B406
 30A8 FF 0E EE          STX    $0EEE
 30AB FF 21 5D          STX    $2150
 30AE FF 21 0E          STX    $210E
                *
 30B1 7E AD 03  QUIT    JMP    $A003
                *
                *
                *PATCHES FOR FCB LENGTH.
                * NOTE THE FOLLOWING
                *ROUTINES ARE LOADED
                *BUT NOT EXECUTED.....
                *
                *MINIFLEX WAS $C0 (192)
                *FLEX 2 IS $140 (320).
                *NEW ROUTINE IS MOVED
                *TO END OF BASIC AND IS
                *THE SAME AS ORIGINAL,
                *($1FB2), EXCEPT 46
                *IS USED INSTEAD OF 376
                *AND 'INC B' USED LATER
                *PROVIDING $146 INSTEAD
                *OF $C6, FOR ALLOCATING
                *FCB SPACE IN SOURCE.
                *
                *JUMP TO PATCH
 1FB2                ORG    $1FB2
 1FB2 BD 24 42       JSR    PATCH
 1FB5 01             NOP
 1FB6 01             NOP
 1FB7 01             NOP
                *


                *PATCH AT END OF BASIC
 2442                ORG    $2442
 2442 86 46  PATCH   LDA A  $46      (WAS $C6 )
 2444 9B 47          ADD A  $0047
 2446 D9 46          ADC B  $0046
 2448 5C             INC B
 2449 39             RTS

 244A           PATEND  EQU    *
                *
                *CHANGE END OF BASIC PTR
 014F                ORG    $14E+1
 014F 4A             FCB    PATEND
                *
                    END    START
      NO ERROR(S) DETECTED


   SYMBOL TABLE:
PATCH   2442    PATEND  244A    QUIT    30B1    START
3000    START1  3003    VN      3002
```

HELP

HELP!

Where can I acquire a copy of Dave Gardner's book "A Companion to Robert H. Ulterwyck's Basic Interpreters"? William R. Hambien mentioned this book in his letter in the January issue. I wrote to SSI at the address given but have received no reply. Looking back through the index of advertisers showed that it is a long time since SSI placed an advertisement in Byte.

Thank you,
D.R. Gaskell
Falkevelen 19
3600 Kongsberg Norway

HELP!

Being a supporter of 68' Micro Journal, I thought you might have a reader who could help me with a problem. I bought the SWTPC Multiuser Board along with their 8K MUB Basic. However I find that it lacks the 'PEEK' and 'POKE' commands that I need to use my Clock Interface an A/d Converter. If any of your readers have added these commands to their copy of MUB Basic or know the patches that I could add to mine, I would like to get in touch with them. This would probably make a good article for 68' Micro Journal!

Thank you,
Steve Powers
Ph. 1-606-236-3538

HELP!

I am having difficulty with a SWTPC MA-2A CPU board. Would some knowledgeable person be willing to check it out for a reasonable fee?

Sincerely,
J. Korman
415 Barberry Ave.
Kalamazoo, MI 49002
1-616-323-0637

HELP!

I have a MEK-02 Kit modified to a 40K Machine running SWTBUG and PERCOM LFD 400 and PERCOM Superbasic. The terminal is a HAL OS 3000. I am interested in using the computer to make a hard copy of baudot code (RTTY) reception from the amateur bands. I would like to be able to run the terminal in Baudot, inputting Baudot via an RS232 interface to the computer and output from the computer to an ASCII printer on port 5. Has anyone written such a program, preferably in machine language?

Sincerely,
Paul E. Phelps WA8ZLJ
111 Division St., 19
King City, CA 93930

HELP!

Although I am writing this with SWTPC V.2 Cassette Basic, I have the TSC Editor and Processor on cassette. However, I do not have a suitable routine in machine language which will allow me to have Hardcopy Output. Does anyone have a routine that will work with my system they would be willing to share with me? I have a SWTPC 6800 with 32K of

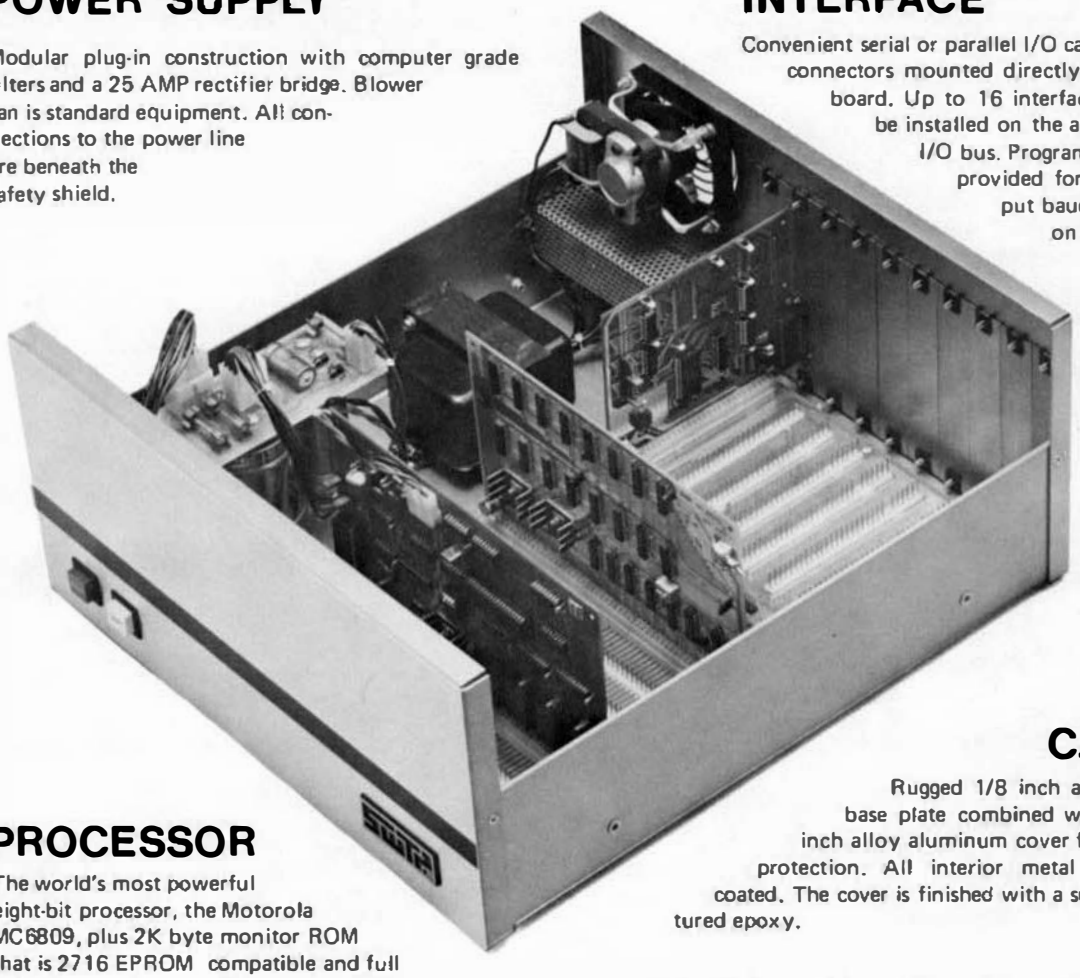
WE HAVE A 6809 FOR YOU

POWER SUPPLY

Modular plug-in construction with computer grade filters and a 25 AMP rectifier bridge. Blower fan is standard equipment. All connections to the power line are beneath the safety shield.

INTERFACE

Convenient serial or parallel I/O cards have DB-25 connectors mounted directly on the circuit board. Up to 16 interface devices may be installed on the address decoded I/O bus. Programming strips are provided for input and output baud rate selection on each port. All outputs are fully buffered.



CABINET

Rugged 1/8 inch alloy aluminum base plate combined with a solid 1/8 inch alloy aluminum cover for unsurpassed protection. All interior metal is conversion coated. The cover is finished with a super tough textured epoxy.

PROCESSOR

The world's most powerful eight-bit processor, the Motorola MC6809, plus 2K byte monitor ROM that is 2716 EPROM compatible and full buffering on all output lines. Built-in multiuser capability, just add I/O cards to operate a multi-terminal system.

MEMORY— You can purchase the computer with either 8K bytes of RAM memory (expandable to 56K), or with the full 56K. The efficient, cool running dynamic memory used in this system is designed and manufactured for us by "Motorola Memory Systems Inc."

PERIPHERALS—The wide range of peripheral hardware that is supported by the 6809 includes: dot matrix printers (both 80 and 132 column), IBM Electronic 50 typewriter, daisy wheel printers, 5-inch floppy disk system, 8-inch floppy disk systems and a 16 megabyte hard disk.

SOFTWARE— The amount of software support available for the 6809 is incredible when you consider that it was first introduced in June, 1979. In addition to the FLEX9 operating system, we have a Text Editor, Mnemonic Assembler, Debug, Sort-Merge, BASIC, Extended BASIC, MultiUser BASIC, FORTRAN, PASCAL and PILOT.

69/K Computer Kit with 8K bytes of memory . . . . . $ 495.00
69/A Assembled Computer with 8K bytes of memory . . . . . $ 595.00
69/56 Assembled Computer with 56K bytes of memory. $1,495.00

SWTPC

SOUTHWEST TECHNICAL PRODUCTS CORPORATION
219 W. RHAPSODY
SAN ANTONIO, TEXAS 78216 **(512) 344-0241**

Think BIG

State of the art "Winchester"type hard disk with a data storage capacity of nearly 16 Megabytes, makes the SWTPC 6809 system the most flexible as well as the most powerful eight-bit microcomputer system in the world. The intelligent controller, using DMA data transfer, makes maximum use of the "Winchester" capability. It is completely compatible with the FLEX9 operating system used on the SWTPC 6809 floppy disk system.

CDS-1 "Winchester" disk drive with controller . .$3,995.00
Cabinet–matching our 6809 computer desk . . . 150.00

SOUTHWEST TECHNICAL PRODUCTS CORPORATION
219 W. RHAPSODY
SAN ANTONIO, TEXAS 78216 (512) 344-0241

memory, a CT-64 connected to an MP-C Card, an AC-30, an Anderson-Jacobson printer going to an MP-LA Card.
Thank you,
Jeffrey M. Craig
Apt. 912-3001 S. King DR.
Chicago, Ill.60616

CLASSIFIED ADVERTISING

SWTPC 6800 Cassette System with FADBUG $450, Call 701-968-4525 or write T. Cartwright, Box 301, Cando, N.D. 58324

OSD 16K memory 2114L's socketed new $260 Call 701-968-4525 or write T. Cartwright, Box 301, Cando, N.D. 58324

Percom LFD-400 and Super Basic for Canadian resident. Canadian Dollars 500. K. Mitadera 514-697-9150 Mon-Fri or 514-694-1643 Weekends & Evenings

Swap Tektronix 545 scope with 4 plugins for 6800 floppy or $700.Jim Caraway, 11 Inwood Circle, Austin, Tx 78746

HELP WANTED - GIMIX needs a good S50 bus technician. If your are interested in steady employment, in the Chicago area, please call 'Bobby' at the telephone number listed below.

NOTE: must be proficient in logical and digital trouble-shooting. Also must have a good understanding of the S50 bus and 6800/09 interfacing. A good telephone personality a plus. Come and grow with us!

GIMIX, 1337 W. 37th Place, Chicago, IL 60609 - AC (312) 927-5510

THE BIT BUCKET

Where all that 'good stuff' falls. Something for everyone.

For the past several months we have been evaluating the 'Osborne' series of accounting programs furnished and written by Great Plains Computer Company, Inc., Box 916, Idaho Falls, Idaho 83401.

The packages reviewed so far include the General Ledger and Accounts Receivable programs. These programs are quite extensive and require a bit of reading to understand all they do. To follow in the near future will be Accounts Payable and Payroll with Cost Accounting. They are merging data file types and when combined make a very complete accounting system of any 6800 or 6809 computer. They are sold only in compiled form (TSC XBASIC with .BAC extensions), however, source listing will be sold seperately for custom modification. I would hasten to add that modification is not recommended as they are complex programs.

Many worthwhile improvements have been made over the original series as distributed by Osborne. Mainly in file structure and dynamic file allocation (FLEX™), this eliminates file reorganization programs and time. Also the keyed-file/ISAM has been enhanced. No sorting routines are used as all data is stored in a sorted sequence.

We originally brought ours up on a TANO 6800 computer with 48K RAM, one serial I/O card and dual 5 inch, single sided mini-drives. We have found no serious flaws todate.

In the last few days we have converted the TANO 5 inch version over to our 56K 6809 computer with double sided, double density 8 inch drives (SWTPC DMF2) and have experienced no difficulty.

As of this time we have not carried our accounting over to this new system. I hope to have our complete accounting running in this package in the near future. As we progress I will report the results of our effort, as many of you have written or called asking about the GPCC 'Osborne' series. The best I can tell you now is that with the results from the sample data files included by GPCC, we have found that it runs well with no errors or bugs that we can find.

For those needing a package that can be customized for small or larger business application, this seems to be a good one. The instructions assume that you have the Osborne manuals, as they are the primary operation instructions. GPCC furnishes a book with each series that entails the changes and improvements they have made, but you will still need the Osborne books to run the system. These may be secured from: Osborne/McGraw-Hill, 630 Bancroft Way, Berkeley, CA 94710.

The price of each package is approximately $295.00 each. This would mean that the average installation cost for a very complete accounting system would be on the order of $1,000.00. Which by most standards is very reasonable.

Our overall evaluation so far rates a 'AAA' from our lab.

OMW

A Basic Tabbing Problem

Some versions of Basic will not support TAB values greater than 127 or 128. This may cause you troubles if you are running your printer at 16.5 cpi and are using 15" wide paper. A cr/lf or cr may be automatically generated before you get to the end of the current line.

The attached program, written using Compureware's Version 8.5 will print 21 columns and then 6 lines of data.

The technique is to handle the data to be printed as strings and not as numerics. You must surpress the cr/lf following the print statements in lines 130 & 250 using the semi-colon. The POS calculation in line 252 will take care of the cr/lf! Be aware that if the last character of the last column 'bumps' the end of the printer some printers will automatically generate another cr/lf. If this happens just delete line 252.

Continuing success to the Journal. An article evaluating an available USDC-Pascal would be very helpful.

Gene Embry
Route 1
Box 151-H
Morrisville, NC 27560

```
0001 : COLUMN.BAS
0002 :
0003 : G. Embry  5/26/80
0004 :
0006 LET W=10
0008 STRING$= W
0010 LINE= 0
0012 LET C=21::# OF COLUMNS
0014 INPUT "PORT ",Q
0020 LET H$="Col. #"
0030 FOR X=1 TO W::Make a blank line
0032 LET X$=" "+X$
0034 NEXT X
0090 PRINT #Q
0099 :
0100 : Print heading
0110 FOR X=1 TO C
0120 LET L$=H$+STR$(X)+X$
0130 PRINT #Q,L$;
0190 NEXT X
0192 PRINT #Q
0198 :
0200 : Print columns of data
0201 :
0210 FOR Y=1 TO 6
0220 FOR X=1 TO C
0230 LET H=INT(RND*100)
0232 LET H$=STR$(H)
0240 LET L$="   "+H$+X$
0250 PRINT #Q,L$;
0252 IF POS>(C-1)*W+1 THEN PRINT #Q
0260 NEXT X
0290 NEXT Y
0299 :
0900 PRINT #Q
0990 END
```

Coming soon will be a monthly (I hope) column devoted to those 68'ers who are non-disk type. This will be headed by Mark Libby, see "A Hobbyist Speaks", May '80, 68 Micro Journal.

The response to his article has been favorable and so I will make some space available for a 'TAPE' type column. The success of this will be YOUR input, for without tape type letters and articles Mark cannot carry the load alone. If you want a tape column each month - THEN GET THE INFO TO 68 MICRO JOURNAL!!!!!

Material concerning hardware as well as software will be needed. A balance of all tape systems (and speeds) will be attempted. Again, this depends on you the user.

So if we have been slack on tape articles (which we have) it is because we have received only a few, as compared to other subjects.

Send all correspondence to:

Mark Libby
3923 Lynncrest Dr.
Cleveland, TN 37311

An additional set of memory locations for those amending FLEX™ 09 for higher disk speeds (see The MPI51/52 Disk Drives, by Dr. Bud Pass), May '80, 68 Micro Journal. These apply to FLEX™ 2.6 as furnished by SWTPC. $DEE5 change 08 to 09, location $DE82 change 18 to 19. I repeat; these changes are for FLEX™ (SWTPC) version 2.6 only, the other locations as in the May issue are for those versions furnished by TSC and not ammended.

Applevalley Day School - PHONE 713-481-3586(after 6 pm)
EXCELLENCE IN PRESCHOOL EDUCATION
3826 ERIE
HOUSTON, TEXAS 77087

SOFTWARE ANNOUNCEMENT - APRIL 28,1980

Applevalley Day School,Inc. is making available our business programs, written in SWTPC Disk Basic,Version 3.0 for 5" floppy disk and the SWTPC 6800 computer. The programs are being offered for use by small service businesses, and can be easily changed to suit special requirements using the hints provided inthe annotated source listings that are furnished. Other documentation provided includes instructions for addapting to the user's system (two to four disk systems accommodated), detail file descriptions, and a tutorial. "dummy" data files are included so that the tutorial can be run before the user enters his data into the files.

The programs were written by Richard C. Cagle, a Texas Proffesional Engineer, and have been in use and continously improved over the last 3 years. The programs are menu selected,use extensive operator prompting, and produce reports on either the terminal or a 40 colum, port #7, parrellel printer. Programs and prices are:

PAYROLL - $25.00
DEPOSIT - $25.00
GENERAL LEDGER - $45.00

The Payroll and Deposit programs write transactions into the Gen. Ledger, which precludes redundant data entry. The Gen.Ledger also includes an automatic amortization/depreciation program and a checkbook reconciliation program. All programs can be purchased as a package for $80.00, which includes 86 pages of documentation.
Both "MINIFLEX" and "FLEX 2.0" versions are available."**
Detailed acenario of program operation is available free. write to:

Applevalley Day School - Software
Richard C. Cagle
11103 Bagepark Lane
Houston,TX,77089

*** The FLEX 2.0 versions still use SWTPC Disk Basic Version 3, which normally runs under MINIFLEX. The software patch to convert to FLEX 2.0 is available free..

'68' Micro Journal
3018 Hamill Road
Hixson, Tennessee

Don:

I enjoy reading your magazine, especially the programs.

Close scrutiny of the "BASIC DECIMAL DOLLAR SUBROUTINE" by John Tarvin reveals a flaw. The dollars are isolated precluding any carry resulting from rounding to the nearest cent. Also, a carry causes the loss of the least significant digit of the cents.

The attached subroutine is offered as an alternate solution. Although, the maximum amount is limited to $ 9,999,999.99.

Sincerely,
Jack D. Johnson
2816 Wood Creek Road
Midwest City, Ok

cc: John Tarvin

```
9000 L=VAL(L$)
9010 L$=STR$(INT(L*100+.005))
9020 L$=MID$(STR$(INT(L/100)),2)+"."+RIGHT$(L$,2)
9030 IF LEN(L$)<6 THEN L$=LEFT$(L$,LEN(L$)-3)+","+RIGHT$(L$,3)
9040 IF LEN(L$)<3 THEN L$=LEFT$(L$,LEN(L$)-3)+","+RIGHT$(L$,3)
9050 L=LEN(L$)
9060 RETURN
```

```
ENTER DOLLAR AMOUNT?  .01                    $ 0.01
ENTER DOLLAR AMOUNT?  .0234                  $ 0.02
ENTER DOLLAR AMOUNT?  .0267                  $ 0.03
ENTER DOLLAR AMOUNT?  .1                     $ 0.10
ENTER DOLLAR AMOUNT?  .12                    $ 0.12
ENTER DOLLAR AMOUNT?  1                      $ 1.00
ENTER DOLLAR AMOUNT?  1.0                    $ 1.00
ENTER DOLLAR AMOUNT?  1.00                   $ 1.00
ENTER DOLLAR AMOUNT?  1.0364                 $ 1.04
ENTER DOLLAR AMOUNT?  987.64                 $ 987.64
ENTER DOLLAR AMOUNT?  17486.01               $ 17,486.01
ENTER DOLLAR AMOUNT?  103467                 $ 103,467.00
ENTER DOLLAR AMOUNT?  9999999.99             $ 9,999,999.99

ENTER DOLLAR AMOUNT?  1.995                  $ 2.00
ENTER DOLLAR AMOUNT?  .0001                  $ 0.00
ENTER DOLLAR AMOUNT?  999999.999             $ 1,000,000.00
ENTER DOLLAR AMOUNT?  1.999                  $ 2.00
ENTER DOLLAR AMOUNT?  .005                   $ 0.01
```

'68' Microjournal
5018 Hamill Rd.
P.O. Box 849
Hixson, TN 37343

Dear Sirs,

Since my previous letter I have been able to have hardcopy from my TSC word editor and text processor system. This is due to the help of two fellow hobbyists who suggested I use a printer driver routine supplied with TSC's MINIFLEX with my cassette software. This letter is being written using the above and as youcan tell it works very well.
The MINIFLEX printer driver routine is:

```
A016 20
A017 01          A04A CE
A018 01          A04B 80
A019 FF          A04C 1C
A01A A0          A04D C6
A01B 35          A04E FF
A01C 37          A04F E7
A01D CE          A050 00
A01E 80          A051 C6
A01F 1C          A052 3E
A020 A7          A053 E7
A021 00          A054 01
A022 C6          A055 39
A023 36
A024 E7          15A4 7E
A025 01          15A5 A0
A026 C6          15A6 9A
A027 3E          15A7 39
A028 E7
A029 01          15C4 7E
A02A 6D          15C5 A0
A02B 01          15C6 16
A02C 2A          15C7 39
A02D FC
A02E 06
A02F 00
A030 FE
A031 A0
A032 35
A033 33
A034 39
A035 00
A036 00
```

The above is for a printer (Anderson-Jacobson) on PORT7.

Sincerely,

Jeffrey M. Crain
Apt.912 - 1001 S. King Dr.
Chicago, IL 60616

R. Lynn Smith
2405 Joseph Drive
Clinton, MD. 20735

May 7, 1980

Dear Mr. Williams:

Well here goes another try an fame and fort ne.

What I've enclosed is a collection of entry points and assorted comments dealing with converting programs between SSB's DOS68 and TSC's FLEX.

As you publish more of those nifty utility and other progr s it seems, at least to me, that there is a real need to get the conversion data for getting from one system to the other in a single place for easy reference. I have tried to do this in the enclosed article. Similar FLEx and DOS68 routines are grouped together with the entry addresses for each listed side by side. Differences between the routines that may affect conversions have been noted. I have also tried to include enough background information on dissimilar routines to give someone without a Programmer's Manual at least a general idea of what the routine does.

The grouping of this information in this form has been useful to me, I hope you and your readers will also find it worthwhile.

Please feel free to use this material as you see fit.

Always enjoy getting the JOURNAL and look forward to seeing some of the software you collected during the contest.

Keep up the GOOD WORk.

Almost forgot. If you have any need to contact me I can be re ched during business hours at (202) 755-4797 or in the evenings at home, (301) 292-3821.

Sincerely,

R L Smith, II

FLEX/DOS68 & DOS68/FLEX
PROGRAM CONVERSION
Similar Routine Addresses

R. Lynn Smith
2405 Joseph Drive
Clinton, Maryland 20735
(301) 292-3821

In a more perfect world we would not have to contend with all the daily minor irritations we have thrust upon on us in this one. The incompatability of the several 6800 disk operating systems is a good example. But then in that perfect world I'm not at all sure any of us would be messing around with anything as trivial as writing software so perhaps we should be content to make the best of what we do have. What this is leading up to is a FLEX/DOS68 conversion listing which just may help in converting all those GREAT programs that you can't use because your TSC FLEX and the program is SSB DOS68 or your DOS68 and the program is for FLEX. Hopefully by grouping similar routines along with entry addresses and register requirements for both systems side by side life can be made just a little more pleasent and the world a wee bit more perfect.

One of my first discoveries when I started gathering information for this article was that most of the user callable routines in the two DOS systems, as you might expect, have quite similar counterparts in each system. They are after all both doing essentially the same job. Unfortunately, there are a few areas where the two programs do not lend themselves very well to easy conversion. Mostly these deal with random access files. It became obvious quite early that random file conversion was beyond the scope of this article (and possibly of the author too). Therefore, what is presented below has been limited to programs using SERIAL access files only. Since most files are this type this should not present most programmers with a serious problem.

Information for this article was taken from Smoke Signal Broadcasting's 'Disk System' For DOS68 version 5.1 ($6000-$7FFF) and Technical Systems Consultants, Inc. FLEX Programmer's Manual For FLEX 2.0.

SYSTEM MONITOR (DOS)

Routines from FLEX and DOS68 system monitors are listed below. Converting most of the routines will involve nothing more then assigning the correct call address. A few are a bit more involved and for these I have tried to give a short description of their differances. Each of the system 'user available' programs has been identified by its functional name, the FLEX address and label followed by the equivalent DOS68 address and lable. Those liste without comments may be considered as being directly interchangeable.

COLD START	$AD00	COLDS	$7280 ZCOLDS
WARM START	$AD03	WARMS	$7283 ZWARMS
DOS RE-ENTRY	$AD06	RENTER	-- --

FLEX Enters FLEX main loop without performing normal Warm Start Initialization. Entry is by a JMP instruction.

INPUT CHARACTER FROM TERMINAL	$AD15	GETCHR	$72C4 ZGET H
	$AD09	INCH	$7289 ZINCH
	$ADDC	INCH2	-- --

FLEX GETCHR is the preferred input routine. Some care should be exercised in converting programs using this call since this routine may also be used to input data from a file. When converting programs from FLEX that call GETCHR users should be aware of this second function and determine that the program is, in fact, being used for a terminal character input. If the routine is used for a file input the address of the FCB will be stored at $AC26-$AC27, otherwise this address will be $0000. The Input Switch Flag, at $AC23, is also checked by this routine. If it is non-zero then INCH2 will be used to input a charter, otherwise INCH is used. GETCHR preserves the B and X-registers upon exit.

INCH2 is the input jump to the system monitor.

INCH is the vector input.

DOS68 ZGETCH is the perferred character input routine in DOS68. The control terminal is called through ZINCH and the routine returns with the input character in the A-register. System parameters are honored by this call.

ZINCH is a jump to the system monitor keyboard input routine. It is assumed that the monitor will not echo the input characters and that the monitor will save the B and X-registers.

OUTPUT CHARACTER TO TERMINAL	$AD18 PUTCHR	$72C1	ZPUTCH
	$ADOF OUTCH	$7286	ZOUTEE
	$AD12 OUTCH2	$72C1	ZOUTCH

FLEX PUTCHR is the perferred output routine. As with input routines PUTCHR is also capable of driving a file when the FCB address is set into $AC24-$AC25. This routine also checks the Output Switch flag at $AC22 and IF found non-zero outputs through OUTCH2 otherwise it uses OUTCH. Registers B and X are preserved.

OUTCH2 is the jump to the system monitor.

OUTCH is the vector output jump.

DOS68 ZPUTCH is the perferred output routine. System parameters are honored and the control terminal port (if ACIA) is checked for a break input. If a break is detected the program will jump to the address stored at YABORT, $730C.

The vector output flag, YOSWT, at $7326 is also checked by this routine. If the flag is non-zero ZOUTEE is called and the character output through the system monitor. If the flag is zero ZOUTCH is called. It is assumed ZOUTCH, if called, has been previously set to an alternate output device.

ZOUTCH is the vectored output. ZOUTCH may be set to jump to an alternate output device's dirver routine. However, it will be reset to ZOUTEE by either a Warm Start or calling ZRESTR.

ZOUTEE is the jump to the System Monitor output.

INPUT CHARACTERS INTO LINE BUFFER	$AD1B INBUFF	$7285	ZLINE1
PRINT AN ASCII STRING	$AD1E PSTRNG	$72A6	ZOUTST

FLEX PSTRNG uses a $04 to terminate the string. It also outputs a carrage return and a line feed prior to feeding the string.

DOS68 ZOUTST uses a $00 to terminate strings.

ALPHANUMERIC TEST	$AD21 CLASS	$729A	ZANCHK
PRINT A CR AND LF	$AD24 PCRLF	$720C	ZCRLF
GET NEXT CHARACTER FROM BUFFER	$AD27 NXTCH	$7297	ZGNCHA
RESTORE I/O VECTORS	$AD2A RSTRIO	$72CA	ZRESTR

FLEX Restores both input and output vectors and resets to zero FILE INPUT and OUTPUT addresses. Preserves the A and B-registers.

DOS68 Restores the output vector. Only the B-register is preserved.

GET A FILE SPECIFICATION	$AD2D GETFIL	$7291	ZFLSPC
FILE LOADER	$AD30 LOAD	$7203	ZLOAD

Loads a binary file into memory. The routines are essentially interchangeable. However, offsets and transfer addresses, if used, must be accounted for.

FLEX The Loader Address Offset is $AC1B-$AC1C. The Transfer Flag location is $AC1D and the Transfer Address is stored at $AC1E-$AC1F.

DOS68 The load offset, YOFSET, address is $732E-$732F. The transfer address flag, YTAFLG, location is $7328 and the transfer address, YTADDR, is stored at $732C-7320.

SET DEFAULT EXTENSION	$AD33 SETEXT	$720F	ZSTEXT

This routine will place a user selected default extension in the FCB pointed to by the X-register provided an extension does not already exist in the FCB. If an extension is present the default will be ignored and the routine will return with the FCB unaltered.

FLEX Upon entry, the default extension must be in the A-register and the X-register pointing to the FCB. Only the X-register will be preserved by this call.

FLEX DEFAULT EXTENSION CODES

0 - BIN	4 - SYS	8 - BAC
1 - TXT	5 - BAK	9 - DIR
2 - CMD	6 - SCR	10 - PR?
3 - BAS	7 - DAT	11 - OUT

DOS68 Data in the B-register (extension code) is transferred to the routine with the X-register pointing to the FCB.

DOS68 DEFAULT EXTENSION CODES

0 - BIN	4 - CTL	8 - TMP
1 - TXT	5 - BAK	9 - ? ?
2 - SAC	6 - DAT	---
3 - BAS	7 - FOR	---

ADD B TO X-REGISTER	$AD36 ADDBX	$72A3	ZADDX

FLEX The B-register is destroyed on exit.

DOS68 The B-register contains the lower order sum on exit.

OUTPUT A DECIMAL NUMBER	$AD39 OUTDEC	--	--

FLEX Enter with X-register pointing to address of 2-byte number. If B-register is non-zero leading zeros will be replaced with spaces. If B is zero, output will start with first non-zero digit.

OUTPUT A HEX NUMBER	$AD3C OUTHEX	$72AC	ZOUTHX
OUTPUT DISK ERRORS	$AD3F RPTERR	$72A9	ZTYPDE
GET A HEX NUMBER FROM LINE BUFFER	$AD42 GETHEX	$72AD	ZGETHN
OUTPUT A HEX ADDRESS	$AD45 OUTADR	$72AF	ZOUTHA
INPUT A DECIMAL NUMBER	$AD48 INDEC	--	--

FLEX Gets a decimal number from the line buffer and stores it, to 16-bit precision, in the X-register.

CALL DOS AS A SUBROUTINE	$AD4B DOCMND	$7200	ZEXCMD

These routines are only similar in function. Conversion, particularly from FLEX to DOS68, may be difficult.

FLEX Allows FLEX to be treated as a subroutine. On entry the buffer must contain a valid command string with the buffer pointer pointing to the first character. Upon exit, DFM (or FMS) error will be indicated by a non-zero status of the B-register.

DOS68 Calling this routine will cause the DOS execute a command string pointed to in the line buffer. Programs called must terminate in an RTS in order for control to return to the original user program. Most utilities do not end with an RTS, but rather JMP to Warm Start. Routines which do not end with an RTS will not work with ZEXCMD.

GET TERMINAL INPUT STATUS	--	--	$72C7 ZSTAT

DOS68 Checks the control terminal's ACIA to see if the input register is full. If it is the Carry bit returns set.

LOOK AHEAD IN LINE BUFFER	--	--	$7288 ZPEEK

DOS68 Looks ahead one character in the line buffer and returns that character in the A-register. The line buffer pointer is not affected.

ABORT COMMAND AND GIVE ERROR	--	--	$7290 ZOIE

DOS68 Prints the contents of the line buffer, a '?' and error message.

OECODE COMMAND NAME AND JUMP	--	--	$7209 ZNAMJ

DOS68 Searches the user command table for a match with the command in the DOS FCB. If a match is made it will call the routine as a subroutine, upon exit, if a match was found the routine exits with the carry clear. If a match was not found upon return the carry will be set.

JUMP TO SYSTEM MONITOR	--	--	$728C ZMON

DOS68 Jump to system monitor: MIKBUG, SWTBUG, SMARTBUG, ETC.

FILE CONTROL BLOCK COMPARISON

In both SSB and TSC disk operating systems data is placed into the FILE CONTROL BLOCK by both the user and the DISK FILE MANAGEMENT system of either DOS. Functions performed by FLEX and DOS68 FCB's are similar and for most programs will involve only slight program code modification to relocate data to the correct FCB location and/or modify the instruction coding. There are a couple areas where files are treated a little differently, so it will not be possible in all cases to nearly 'plug in the correct numbers'. We'll try to point out these as they come up.

DOS68 uses two sizes of FCB's. Earlier versions, before random files were introduced, used a 166 byte FCB. In later versions programmers using DOS68 have a choice, sequential only files still need only 166 bytes or random access files which require 320 bytes for the FCB. FLEX 2.0 files require a 320 byte FCB for all file types. Since only sequential files are being considered at this time either size FCB may be used in converting FLEX programs to DOS68.

The FCB's of both systems are quite complex and capable of transferring a great deal of data to and from the DFM. A complete discussion of all aspects of convertion between FLEX and DOS68 would take considerably more space and time then this article will permit. Fortunately, most programs utilize only a very few of the features offered by these systems. The limited presentation given below, dealing with sequential files, should be sufficient to convert all but the most involved file handling programs.

FUNCTIONAL COMPARISON

FUNCTION CODE (XFC -- OPERATION CODE)

FLEX BYTE $00
DATA $00 Read or Write from/to file.
If file is open to read, a call to FMS will bring next character back in the A-register. If open to write the character in the A-register will be placed in the file with a call to FMS. Note this is somewhat different than the method used by DOS68.

DATA $01 Open file to Read.
DATA $02 Open file to Write.
DATA $04 Close file.

DOS68 BYTE $00
DATA $01 OSOWW -- Open sequential Write file.
DATA $02 OSWRIT -- Write into a sequential file.
Must be placed in FCB prior to calling DFM to

```
                write to a file.
        DATA $03  OSWC -- Close a sequential Write file.
        DATA $04  OSO4R -- Open a sequential Read file.
        DATA $05  OSREAD -- Read from a sequential file.
                  Must be placed in FCB prior to calling DFM
                  when Reading a file.
        DATA $06  OSRC -- Close a sequential Read file.

ERROR CODES (XES -- ERROR STATUS RETURNED TO CALLER)

                  This byte is used by both systems to return
                  DFM (FMS) errors.

FLEX    BYTE $01
        DATA $08  Read past END OF FILE.

DOS 8   BYTE $01
        DATA $06  EEOF -- END OF FILE.

ACTIVITY STATUS (XFS -- FILE STATUS)

FLEX    BYTE $02
                  Used by FLEX to report back the current file
                  status. A $01 if the file is open for Read and
                  a $02 if it is open for Write.
DOS68   BYTE $0D (13)

        Returns current file status.  A $01 for sequential Read,
        a $02 for sequential Write and a $03 for random access.

SPACE COMPRESSION FLAG (XFT -- FILE TYPE)

FLEX    BYTE $3B (59)
                  Allows space compression to be used on ASCII
                  ty  files. A $FF in this location turns off
                  compression (for binary files) and a $00
                  through $7F (positive) value turns the
                  compression on.
DOS68   BYTE $0C (12)
                  FTX -- Four types of files are allowed. A $01
                  for sequential ASCII compressed, a $02 for
                  sequential binary, a $04 for byte mode random
                  and a $05 for record mode random.

                     DISK FILE MANAGEMENT

Calls to either system's DFM are are essentially the same. Each
DFM (or FMS) has three user calls available,

        1.  Initialization
        2.  I/O processing
        3.  File management closure

Neither the initialization or closure routines require a FCB. In
essence these are global calls which work on the DFM as a whole.

INITIALIZATION

FLEX    Entry $B400  FMS Initialization.
        Initalization normally will be taken care of by the  DOS.
        User programs should not be required to call this
        routine.

DOS68   Entry $7780  CDFM -- Initialization Entry Point.
        SSB's manual recommends this call be made to reset
        internal DFM flags and clear any open files.

I/O DFM PROCESSING

FLEX    Entry $B406  FMS Call.
        Upon entry the X-register must point to a properly
        formatted FCB.  Depending on the function being performed
        data will be transferred from the A-register or placed in
        it.  Errors are noted by the state of the zero condition
        code bit upon return.

DOS 8   Entry $7786  DFM -- I/O Service Request Entry Point.
        Entry requirements and exit register status is the same
        as FLEX I/O DFM.

CLOSE ALL OPEN FILES

FLEX    Entry $B403  FMS Close.
        Calling this routine closes all open files.

DOS68   Entry $7783  CDFM -- Closing Entry Point.
        Same functions as in FLEX.
```

It has not been my intention to make this a substitute for the FLEX or DOS68 system manuals. At best it is hoped that this will provide some assistance to those who wish to convert programs but do not have both manuals available to them to dig the conversion data out for themselves.

Most of the information presented has been taken directly from the FLEX and DOS68 instruction manuals and therefore is assumed correct.

ROY G. CALDWELL
13656 BORA DR.
SANTA FE SPRINGS CA. 90670

DON WILLIAMS, EDITOR
'68' MICRO JOURNAL
3018 HAMILL RD.
PO BOX 849
HIXSON, TENNESSEE, 37343

DEAR DON:
HERE IS A SHORT BASIC SUBROUTINE THAT READS THE DATE FROM DOS68

THE VARIABLE X9 IN LINE 9905 MAY BE CHANGED FOR OTHER VERSIONS OF DOS68. 51X
IE. FOR DOS 8.51A USE X9=45872
AND FOR OS68.51 USE X9=29488
NOTE: THESE NUMBERS WERE NOT TESTED BUT SHOULD WORK.

```
9900 REM SUBROUTINE TO READ THE DATE STRING FROM DOS68.51C
9901 REM USES VARIABLES A9, I9, X9
9902 REM DATE IS RETURNED IN D$
9905 LET X9=54064    : REM DATE STRING IN DOS68.51C
9910 FOR I9=0 TO 14
9920 LET A9=PEEK(X9+I9)
9930 IF A9=0 THEN 9950
9940 LET D$=D$+CHR$(A9)
9950 NEXT I9
9955 REM THE NEXT LINE TEST THE STRING TO SEE IF
9956 REM WE GOT THE DATE OR JUST GARBAGE
9957 REM MID$(D$,12,2) RETURNS THE FIRST TWO DIGITS
9958 REM OF THE YEAR WHICH SHOULD BE "19"
9960 IF MID$(D$,12,2) = "19" THEN 9980
9970 INPUT "ERROR READING DATE.  ENTER DATE";D$
9980 RETURN
9999 END
```

Roy

MORIZONTAL COLOR BAR SUBROUTINE FOR THE MOTOROLA MICRO CHROMA 68

Tom J. Ramaco
1505 Magnolia Drive
Salisbury, MD 21801

I am submitting an example of a subroutine (MLINE) that generates horizontal color bars for the Motorola Micro Chroma 68. MLINE saves and restores all registers. However, you may wish to store the X register in some locations other than 0 and 1. The technique of passing arguments by value can also be applied to your own 6800 subroutines.

The calling sequence is:

```
        JSR  MLINE      CALL SUBROUTINE MLINE
        BRA  *+5        RETURN ADDRESS
        DB   nn         LINE NUMBER (0 thru F)x
        DB   nn         COLOR/PATTERN
        DB   nn         NUMBER OF COLUMNS (0 thru FF)x
```

The subroutine arguments are defined as follows:

LINE NUMBER
Bits 3-0 - select line number
0 = the top line on the TV display (1st line)
F = the bottom line on the TV display (16th line)

COLOR/PATTERN
Bit 7 = 1 selects SEMIGRAPHICS FOUR mode. Each picture element (PIXEL) is divided into four equal parts where the luminance of each part is controlled by bits 3-0.

Bits 6-4 - select the color of the illuminated parts.
000 - green
001 - yellow
010 - blue
011 - red
100 - buff
101 - cyan
110 - magenta
111 - orange

Bits 3-0 - luminance control
0000 - black rectangle
$L_3L_2L_1L_0$ - illuminate the corresponding part of the PIXEL defined by the diagram below.

L_3	L_2
L_1	L_0

NUMBER OF COLUMNS
Bits 7-0 - select number of columns

SUBROUTINE MLINE

HEX ADDRESS	HEX DATA	LABEL	OPCODE/OPERAND	DESCRIPTION
0200	36	MLINE	PSHA	SAVE THE A REGISTER
0201	37		PSHB	SAVE THE B REGISTER
0202	DF 00		STX 0	SAVE X REGISTER IN LOC. 0,1
0204	30		TSX	MOVE SP INTO X
0205	EE 00		LDX 0,X	X = RETURN ADDRESS
0207	A6 02		LDAA 2,X	A = LINE NUMBER
0209	5F		CLRB	B=0
020A	84 0F		ANDA #$F	IGNORE BITS 7-4
020C	27 0A		BEQ AA	TEST FOR LINE ZERO
020E	48		ASLA	NO - COMPUTE
020F	59		ROLB	2 * LINE NUMBER
0210	48		ASLA	
0211	59		ROLB	4 * LINE NUMBER
0212	48		ASLA	
0213	59		ROLB	8 * LINE NUMBER
0214	48		ASLA	
0215	59		ROLB	16 * LINE NUMBER
0216	48		ASLA	
0217	59		ROLB	32 * LINE NUMBER
0218	CD D0	AA	ADDB #$D0	COMPUTE DISPLAY ADDRESS
021A	F7 F3 CC		STAB NEXTBY	MOVE CURSOR POSITION
021D	B7 F3 CD		STAA NEXTBY+1	TO DESIRED ROW
0220	A6 03		LDAA 3,X	A = COLOR/PATTERN
0222	E6 0A		LDAB 0,X	B = NUMBER OF COLUMNS
0224	27 09		BEQ INTN	RETURN IF ZERO
0226	FE F3 CC		LDX $F3CC	X = CURSOR POSITION
0229	BD F8 03	AGAIN	JSR $F803	OUTPUT PIXEL
022C	5A		DECB	B = B - 1
022D	26 FA		BNE AGAIN	CONTINUE UNTIL B = 0
022F	DE 00	INTN	LDX 0	RESTORE THE X REGISTER

```
0231    33          PULB        RESTORE THE B REGISTER
0232    32          PULA        RESTORE THE A REGISTER
0233    39          RTS         AND RETURN
```

This subroutine is relocatable and can be moved to any RAM address other than 0 or 1 without having to change any of the hex data values.

KLINE can also be used to position the cursor at the beginning of any line on the display by specifying the appropriate line number with a column count of zero. For example, to home the cursor, use line number 0 with a zero column count.

You can use this subroutine as a building block towards generating a histogram with horizontal color bars. You might want to try and modify KLINE to draw vertical color bars.

If you want to selectively erase or fill specific lines on the TV display with ascii characters, then replace the color/pattern byte in the calling sequence to KLINE with the ascii character to be output. For example, use 20, to erase characters on the line(s) specified. You are actually entering the alphanumeric display mode whenever bit 7 of the color/pattern argument equals zero.

Note that the SEMIGRAPHICS FOUR mode generates 16 lines with 32 columns per line. Therefore, a call to KLINE with a column count greater than 63, will wrap around to the next line. The actual resolution of the SEMIGRAPHICS FOUR mode is 32 x 64 since each PIXEL is divided into a 2 x 2 matrix as illustrated under the color/pattern description.

An example of a call to KLINE that will draw a solid orange bar on the last line of the TV display is given below:

```
ADDRESS   DATA          LABEL   OPCODE/OPERAND      DESCRIPTION

0100
0100      BD 02 00      MAIN    JSR  KLINE          DRAW A HORIZONTAL COLOR BAR
0103      20 03         *       BRA  *+5
0105      0F                    FCB  $0F            LINE NUMBER
0106      FF                    FCB  $FF            ORANGE/PATTERN
0107      20                    FCB  $20            32 COLUMNS
0108      7E F8 21              JMP  $F821          RETURN TO TVTOO MONITOR
```

References

Motorola MC6847 Video Display Generator Spec. Sheet

Micro Chroma 68, The New "Bug" from Motorola TVTOO 1.2

available from
Motorola Microcomputer Applications
PO BOX 20912 Attn: MicroChroma 68
2501 W Dimmitt Blvd.
Austin, TX 78720

Don Williams, Editor
68 Micro Journal
5018 Hamill Rd.
Hixson, Tennessee 37343

12 May 1980

Dear Don,

First I want to congratulate you for your fine magazine. At this side of the ocean it is sometimes difficult to keep in touch with what is going on in the 6800 world, and your magazine is certainly an excellent way of doing it.

I enclose material on the USR function in the TSC Basic which may prove useful to some of your readers. However, I would be grateful you could let me know if it is not suitable for publication.

Using the USR function in BASIC ? It may be a bit of a hassle, specially if you are not totally familiar with Flex and its idiosincrasies. It needs a reasonable amount of fiddling around with PEEKs and POKEs, and, worst of all, it wont work if you dont have that assembler routine loaded somewhere in your memory.

The advent of the 6809 makes possible to use auto-relocating programming as a solution. You write the assembler code, then you relocate it to some place in storage where BASIC won't destroy it, set up the required environment and off you go. This will sound either very simple or very difficult to you, depending on how much FLEX and assembler you know. Well, it is not difficult - in fact it is quite simple, but it involves some degree of high precision work, and the result is rewarding because you end up with quite a flexible tool to put together your assembler subroutines.

Auto-relocating programming is writing assembler programs which can be loaded anywhere in memory and work all the same. In other words, once it is in core you can move it around without having to pay attention to relocation of addresses and other variable information. In practical terms, it means that:

1. All references to labels should be made using relative addressing. For example, one should use LEAX LABEL,PCR as opposed to LDX LABEL because it will generate a displacement in relation to the current PC register, while the LDX will generate a real 16 bit address which unless relocated by some technique will require that label to be actually loaded at the generated address (otherwise the program will not work).

2. Local constants and variables should be allocated in the stack and not by use of FCCs, FCBs etc. For example, if you use 3 16-bit areas in your program, they should be equated to displacements from the stack base as follows:

```
VAR1    EQU   $0
VAR2    EQU   $2
VAR3    EQU   $4
SIZE    EQU   $6
```

When the program starts, you can reserve space in the stack for those variables by issuing the instruction

```
        LEAS    -SIZE,S
```

and reference your variables via displacements from the S register:

```
        ASGN    #VAR3,S
        LDY     #VAR1,S
```

No variable and area labels will have to be relocated. When exiting the program, it is sufficient to do an

```
        LEAS    SIZE,S
```

to restore the stack to its previous position.

3. Do not use jumps or JSRs, but LBSRs and long branches instead. This will generate 16-bit displacements from the current PC as opposed to 16-bit addresses, guaranteeing that no relocation is required.

4. It is ok to use the absolute addresses of the FLEX routine vectors and entry points. After all, they are not that relocatable anyway.

Now for the USR function itself. The TSC BASIC requires that the address of the user-declared USR function is contained in the two bytes preceding the FLEX end of memory. In FLEX9 this is location BFFF hex, and BASIC maintains it in slots CC2B-CC2C. That is, if you PEEK locations CC2B-CC2C hex you get the current FLEX end of memory as seen by BASIC. So if you want to find the end of memory location (from now on I will call it MEMEND), the following BASIC sequence does it:

```
M1=HEX("CC2B")
M2=HEX("CC2C")
TP=256*PEEK(M1)+PEEK(M2)
```

Variable TP now has the floating point equivalent of the MEMEND address, and can be used in any POKE statement. So if you have ATe the address of your user assembler subroutine in variable ZZ, you can store it in MEMEND -2 and MEMEND-1 with the following sequence:

```
Z1=INT(ZZ/256)      : REM equivalent of left byte of  routine address
ZZ=ZZ-256*Z1        : REM equivalent of right byte of address
AD=TP-2             : REM AD now has MEMEND-2
POKE AD,Z1          : REM store left byte of routine address
POKE AD+1,ZZ        : REM store right byte of routine address
```

Now you can call USR. The trouble is, how do you load your assembler subroutine and how do you get its address in ZZ? BASIC starts at address 0000 and will use up to the end of memory if you let it. The solution is to load the assembler subroutine beyond the end of memory, so that BASIC wont overlay it. In other words, we move the MEMEND address downards so that there is enough space beyond MEMEND to load our routine.

If we only knew how many words we need... Unfortunately we dont, as it may take a whole pass through all of the BIN file of your assembler program to find out its size. The solution I adopted was to include a small relocator within my assembler subroutine so that when I wanted to load the USR assembler I called it as a program (for example ***1.SUBROUTN.BIN). The entry point is within the relocator which relocates the actual subroutine machine code to high core (beyond MEMEND). It is then possible to load BASIC and run any program that uses that USR subroutine.

The relocator is a 12-line assembler sequence that computes the size of your subroutine, bumps MEMEND down so that the subroutine fits beyond MEMEND, relocates the subroutine's machine code starting at MEMEND+1 and exits. It assumes that the subroutine itself is contained between labels TOP and BOTTOM in the same module, and that the subroutine entry point is actually at label TOP. The following listing shows a sample user subroutine with the relocator in front(the subroutine gets one character from the CT-82 without echoing it back and passes it directly to BASIC without requiring the user to type a carriage return). The relocator starts at ENTRY, and first computes the new MEMEND by subtracting BOTTOM-TOP from it. This will make sure that enough space is left from the new MEMEND+1 location for the assembler subroutine. It then stores the address of MEMEND+1 at MEMEND-2 and MEMEND-1 which is where BASIC expects to find the start location for the USR subroutine. The relocator then proceeds to move the object code between labels TOP and BOTTOM to the space from MEMEND+1 on by doing a straight copy word by word. It finally returns control to FLEX.

```
MEMEND  EQU   $CC2B          * flex end of memory slot
RENTER  EQU   $CD06          * dos reentry address
ACIADR  EQU   $E8            * BASIC slot for ACIA address
*
*       this section relocates the actual program
*       to high memory (beyond memend) and reallocates
*       memend. the program being relocated starts at
*       label TOP and ends at label BOTTOM. Anything
*       can be inserted between those two labels as
*       long as there is enough memory available and
*       the program is auto-relocating. It is advisable
*       that TOP-BOTTOM is an even number as the relocator
*       moves word by word rather than byte by byte.
*
ENTRY   LDX   MEMEND         . get end of memory address
        LEAX  TOP-BOTTOM,X   . adjust for routine size
        STX   MEMEND         . and store back
        LEAY  1,X            . get address of routine start
        STY   -2,X           . and store in MEMEND-2 for
                             . the BASIC interpreter
        LDU   #(BOTTOM-TOP-1)/2 . compute routine size (words)
        LEAX  TOP,PCR        . point to start of routine
XFER    LDD   ,X++           . get a word
        STD   ,Y++           . relocate to high memory
        LEAU  -1,U           . decrement count
        BNE   XFER           . and go do next word
        JMP   RENTER         . done, back to flex
*
*       this section is the program to be relocated. It must
*       be wholly included between TOP and BOTTOM and must
*       fit within the required or available memory. the
*       program shown below gets a character from the CT-82
*       and passes it to BASIC via the USR facility so that
*       single character input is achieved without the need
*       to press return.
*
TOP     LDX   ACIADR         . get ACIA base address
NTOP    LDA   ,X             . Get contents of control reg
        LSRA                 . move right bit to carry
        BCC   NTOP           . loop again until set
        LDA   1,X            . now get the character
```

```
        LDX   MEMEND          . get memory end
        STA   -4,X            . store as parameter for USR
        RTS                   . and return to caller
        FDB   $0              . just in case, pad with 0s
BOTTOM  FDB   $0              .
        END   ENTRY           . and define the entry point.
```

If this program is started at $C100, it may be executed from within BASIC via +SUBROUTN.BIN (for example), or else from FLEX (+++SUBROUTN.BIN). This will load the user subroutine and prepare it to be called via USR from BASIC programs. The following sequence is a BASIC subroutine to prepare the environment for a USR reference. A GOSUB 32000 should precede the actual call to USR.

```
32000 X1=MEK('CC2B')                  : REM top of memory in BASIC
32010 M2=MEK('CC2C')
32020 TP=256*PEEK(M1)+PEEK(M2)        : REM get memend address to TP
32030 Z2=1P+1                         : REM memend+1 is the start of the routine
32040 Z1=INT(Z2/256)                  : REM isolate left byte
32050 Z2=Z2-256*Z1                    : REM isolate right byte
32060 AD=1P+2                         : REM this should be where routine address
                                        goes
32070 POKE AD,Z1                      : REM    left byte of routine address
32080 POKE AD+1,Z2                    : REM    right byte also inserted
32090 RETURN                          : REM user can now call USR subroutine
```

Lines 32000 through 32030 get the address of MEMEND+1 (which is by convention where the USR assembler subroutine starts) and puts it in variable Z2. Lines 32040 and 32050 isolate the two bytes (left and right) of the address in variables Z1 and Z2; this is necessary because TSC basic does not have the double length PEEK and POKE routines. Line 32060 computes MEMEND -2 (this is where the routine start address should be stored for USR to work) and sets variable AD with it. Finally lines 32070 and 32080 write the routine start address into MEMEND-2.

If this routine is executed (for example with a gosub) before every reference to USR, it will work ok. However, if the MEMEND environment is not changed within the program or afterwards, it is enough to execute lines 32000-32060 only once, say, at the beginning of the pro ram. Lines 32070 and 32080 may need to be executed every time, specially if AD changes or Z2 changes. I would suggest calling the whole routine every time (the overhead is not that big, and in the case of the one-character input routine is hardly noticeable.

A sample program to use the above BASIC subroutine follows. It reads one character from the CT-82 and prints its ASCII equivalent:

```
10 PRINT 'ENTER CHARACTER: ';
15 GOSUB 32000                        : REM prepare to read
20 AA=USR(0)                          : REM read the character
25 A1=INT(AA/256)                     : REM isolate the character
30 PRINT CHR$(A1);': ASCII ';A1       : REM print character and its ascii
40 IF A1=13 THEN STOP                 : REM carriage return ends program
50 GOTO 10                            : REM repeat
```

It should not be necessary to type a carriage return after every character. In fact, the carriage return will terminate the program. Note that the assembler USR routine must be run before this program is run so that the actual character read code (from label TOP to label BOTTOM in the assembler listing) is loaded in high memory (beyond MEMEND) and the corresponding USR environment set. Note also that the USR subroutine included as example does not echo the character back to the CT-82 so that it will print the ASCII value of any input character, be it data or control.

In summary, the sequence to integrate a new USR subroutine is:

1. Include the subroutine within labels TOP and BOTTOM in the relocator module, replacing the example routine given);
2. assemble it, for example, calling it SUBRTN.BIN;
3. before running your basic program, run SUBRTN.BIN to move the routine to high core. It is possible to include it in your STARTUP sequence;
4. Use the BASIC subroutine (32000 etc.) and the sample call sequence to actually access the USR subroutine.

Good luck...

Alberto Caesar Moreira
22 The Paddock
Chalfont St Peter
Bucks SL9 0JQ
England

M. S. Ritchie
2806 N.W. 54th Ave.,
Gainesville, Florida 32601

Dear Mr. Ritchie:

Your 'help' cry in the May, 1980 issue of '68 Micro Journal has touched my heart.

The enclosed possible solutions for your CT-64 problem are prepared in the form of an article -- a copy of which I'm sending along to Don Williams, in case other people may be suffering from the same problem. With a regard of one other in the world, I now conclude that I wasn't the only one with this problem!

With regard to Problem 2, I'm not sure I can help. My CT-64 is getting along OK with FLEX 1.0 for the OMNI-1 8" disk, and got along fine with most of the early MSI software. Much 6800 firmware and software insists that $1S (^W^R) means "erase to end of line" -- and you kinda hafta go along. Similarly, much 6800 software uses $1B (CTRL/^) for home-up and $1G (CTRL/W) for erase-to-end-of-page. There aren't any additional conventions regarding control characters that I know of (for reverse video, scrolling, page-swap, non-blink, etc.), so you bet these up to suit yourself. The normally-used characters (CAN, ESC, LTx, BS, LF, CR, VT, HT) should be avoided for funny functions, of course, since they are already assigned standard functions in either the CT-64 or in much standard software. Uniterm uses CTRL/O as the default BS character in his early BASIC's, so that one should also be avoided for CT-64 functions. Otherwise the MMSI-FLEX and CT-64 manuals should tell you everything you need to know.

Oh Okidata, I'll pass. Maybe somebody else can help.

Yours truly,

Geoffrey A. Glass

c: Don Williams
'68' Micro Journal

5248 S. W. Dosch Rd.,
Portland, Oregon 97201

May 5, 1980

Intermittent doubling-up of letters when using the SWTPC CT-64 terminal system can usually be traced to one of four causes -- two of which seem obvious and are usually not the case, and two of which are less obvious but possibly more common.

The first of the more obvious causes is true key-bounce, from weak or contaminated contacts in the keyswitch. This cause is not very probable, as the GI AY-5-2376 encoder chip in the KBD-5 keyboard has a fairly effective debounce circuit. If true keybounce is the problem, an alternative to repairing or replacing the bad keys is to increase the debounce timing resistor (R1) value from 680 k to 1 M or so. Sooner or later, the bad key or keys will have to be replaced or repaired. (Repair requires a steady hand, good eyesight, a magnifier and a pair of long, skinny tweezers. Lacking any one of these or a lot of patience, the owner should elect replacement.)

Another low-probability cause is the repeat-initiation timing being too short, as from a diminished value for C7 (220 uf). The actual delay can be evaluated by deliberately holding down a key and observing the delay before it starts repeating.

A higher probability is an "encoding strip" (stiffener) broken loose on the bottom side of the keyboard. A sharp blow to the keyboard, as from kids banging on it or from a falling object, can break a solder-joint. Since these strips do some of the bussing under the keyboard, an intermittent can cause severe "bounce" when the keyboard flexes under the stress of keystrokes. The clue here is that the "bounce" will occur for

a specific group of keys (like O,L,+ and ;) associated with a single column of the encoding matrix. In my CT-64, the damage was caused by an instrument cart going over a bump, bouncing the CT-VM monitor down onto the keyboard. It was months before I found the microscopic crack in an encoding-strip solder-joint that was causing all my problems.

The last probable cause only shows up in systems in which the keyboard input character is echoed back to the terminal by the computer, and the CT-64 ECHO switch is kept down. The clue here is that the doubling-up only occurs in the display, and the computer never sees doubled characters. The problem is in the latching ECHO keyswitch, which may chatter as other keys are struck. If the ECHO line pops high for just a microsecond or so, the terminal control circuits will respond with a local display of the keystroke, and later with the computer-echoed character. Clue: If the problem never occurs in 'local' (ECHO switch up, RCVE/XMT switch down), look to repairing or replacing the ECHO switch -- maybe with a toggle switch on the panel in one of the locations provided, or with a new keyswitch. For a quick-fix, hang a capacitor on the ECHO line -- a microfarad or two -- to swamp out any momentary "opens". I presently have a chunk of 14-gauge wire physically holding the switch down -- but still have occasional problems, and will continue to have them until I take my own advice!

We have received a program written end sold by Dale A. Chamberlain, 7701 meadowlark Dr., Godfrey, IL 62035, which is a BASIC cross reference utility.

It is current only available for TSC BASIC however, we understand that it is being updated for XBASIC also. We have tried it on XBASIC (see example) as well es BASIC and found it useful and a worthwhile addition to the BASIC programmers set of utilities.

It will work with both 6800 end 6809 BASIC source programs but is written to execute on e 6809 machine. The selling price is about $25.00.

It is also being offered by the Computer Systems Center, Hazelwood, MO (see advertisement this issue). Interested readers should contact the above store or the author for complete details. We will run a review after we receive the XBASIC version, which seems to be the most opular of the two versions. But to say the least, the fellows in our lab who program in BASIC say it is a fine software tool end well worth the money. They all agree it is a AAA rated software offering.

DMW

```
10 REM FIND MISSING LINES
20 REM ALL FIRST LINES MUST HAVE A '/'
30 REM THEREFORE THE LABEL # RECORD SHOULD
40 REM START WITH A '/'
50 ON ERROR GOTO 280
60 INPUT "FILE NAME - NO .EXT",X1$
70 F$=X1$+".TXT"
80 OPEN OLD F$ AS 1
90 N%=0
100 INPUT LINE #1,Q1$
110 W1$="/"
120 I%=1
130 P%=INSTR(I%,Q1$,W1$)
140 IF P%=0 THEN GOTO 240
150 FOR X%=1 TO 5
160 ON ERROR GOTO 290
170 INPUT LINE #1,Q1$
190 NEXT X%
200 N%=N%+6
210 GOTO 100
220 CLOSE 0:CLOSE 1:POKE 52233,10
230 END
240 PRINT CHR$(7):PRINT
250 PRINT "**ERROR** OCCURED JUST PRIOR TO THIS RECORD!
    - ";N%;Q1$
260 PRINT:PRINT"NOTE: CORRECT ERROR IN EDIT AND THEN RERUN!!"
270 GOTO 320
280 IF ERR=4 THEN PRINT "NO SUCH FILE! - TRY AGAIN"
290 IF ERR=8 THEN PRINT"NO ERRORS FOUND!":GOTO 220
300 CLOSE 1
310 GOTO 60
320 PRINT:N%=N%+1: FOR X%=1 TO 6
330 N=N+1
350 INPUT LINE #1,V1$
360 PRINT N%,V1$
370 NEXT X%
380 GOTO 220
```

```
CROSS-REFERENCE OF PROGRAM ERR          05/20/80

"**ERROR**                  250
".TXT"                      70
"/"                         110
"FILE NAME                  60
"NO ERRORS                  290
"NO SUCH F                  280
"NOTE: COR                  260
01$                         170
F$                          70 80
I                           120 130
IN                          130
N                           90 200 250 320 330 360
P                           130 140
Q1$                         100 130 250
R                           130
ST                          130
V1$                         350 360
W1$                         110 130
X                           150 190 320 370
X1$                         60 70
60 CALLED BY                310
100 CALLED BY               210
220 CALLED BY               290 380
240 CALLED BY               140
280 CALLED BY               50
290 CALLED BY               160
320 CALLED BY               270
```

40 TRACK FORMAT FOR MINI-FLEX USING WANGCO DRIVES

THIS MAY BE DONE BY MAKING THESE CHANGES IN THE NEWDISK.CMD.

027F FROM 23 TO 29
02C7 FROM 23 TO 29

02FE FROM 64 TO D0
0307 FROM 22 TO 28
034E FROM 23 TO 29

NOW SAVE NEWDISK,200,5FB,200. YOU CAN NOW EITHER REPLACE THE NEWDISK.CMD WITH THIS BINARY FILE OR RENAME IT AS YOU CHOOSE.

ONE CHANGE 'MUST' ALSO BE MADE IN THE DOS SYSTEM. THE CONTENTS OF MEMORY LOCATION 7A75 MUST BE CHANGED FROM 22 TO 28.

IF THE DOS.SYS IS LOCATED IN CONSECUTIVE SECTORS STARTING AT 0101 ON THE DISK, BYTE 131 ON SECTOR 0112 MUST BE CHANGED FROM 22 TO 28.

FOR THOSE THAT DO NOT HAVE A UTILITY TO MAKE THIS CHANGE, A SHORT PROGRAM FOR IT IS IN FIG. 1. PUT THE DISK IN DRIVE '0', USE THE MON COMMAND AND HAND LOAD IT. SET M A048 TO 0100 AND HIT 'G'. IF THE PROGRAM FAILS TO READ OR WRITE THE SECTOR PROPERLY THE LETTERS 'NG' WILL BE PRINTED. OTHERWISE THE PROGRAM WILL RETURN TO THE DOS SYSTEM.

TO CHECK YOUR RESULTS BOOT UP THE SYSTEM AGAIN AND CHECK MEMORY LOCATION 7A75. THIS LOCATION SHOULD NOW READ 28.

NOTE: THE DISK MUST BE IN DRIVE '0'! ALSO, TO INSURE PROPER OPERATION NEWDISK A DISK, COPY THE DOS.SYS TO IT SO THAT IT WILL START AT SECTOR 0101. IF NO BAD SECTORS ARE REPORTED BETWEEN SECTOR 0101 TO SECTOR 0112 ON THE NEWDISK OPERATION, NO PROBLEMS SHOULD BE ENCOUNTERED.

DOS PATCH PROGRAM
LENGTH 47 BYTES

```
0100  CE 01 12        0118  A7 00
0103  FF 77 5E        011A  8D 03
0106  CE 77 40        011C  7E 71 03
0109  86 09           011F  BD 73 06
010B  A7 00           0122  26 01
010D  4F              0124  39
010E  A7 03           0125  CE 01 2D
0110  8D 0D           0128  BD 71 10
0112  86 28           012B  20 EF
0114  A7 B5           012D  4E 47
0116  86 0A           012F  04
```

FRANCIS E. VAN HORN
418 ESTES STREET
MURFREESBORO, TN
37130

P. O. Box 4026
Bellevue, WA 98009

May 13, 1980

'68' Micro Journal
6131 Airways Boulevard
Chattanooga, Tennessee 37421

Gentlemen:

Does any reader seriously doubt that MIKBUG has set the 6800 business back 2-3 years compared to the competition? Its non-grouped entry points, and placement of ports and scratch memory dead center in the memory map, have been a serious obstacle to development of larger, more capable systems.

It has one redeeming feature, however. It can serve as a horrible example when the time comes to devise monitors for the 6809 and 68000.

Would '68' Magazine assume leadership to try to set some minimal standards? Someone other than hardware manufacturers must do this to provide the necessary objectivity.

As a starting point, we would suggest the following for the 6800:

1. Reserve the entire block from $F000 to $FFFF for the monitor and other EPROM. This should give reasonable freedom to use this space, with provision for startup vectors.

2. Make all essential or frequent entries to monitor subroutines through a vector table starting at $F000. EXBUG could serve as a model for this, but should be expanded.

3. Assure that all addresses are fully decoded (except the high end of the "F" block where interrupt vectors may dictate otherwise).

4. Place ports above $EF00 in order to leave maximum space for contiguous memory.

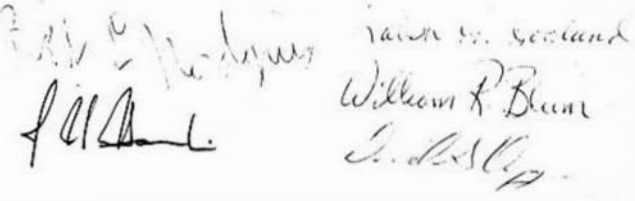
5. If flexible addressing is provided, the above considerations could be adhered to as long as RAM has available to it a maximum contiguous block from zero up to, say $EF00.

6. Do not put disc operating systems in low memory.

7. Place scratch RAM for operating systems at as high an address as possible.

8. Furnish source listings of monitors to permit easy adjustment of the above parameters, or provide some means to adapt to different memory sizes.

The above is hardly perfect, but should serve as a guide to avoid some of the deficiencies in our present systems.



Ralph Roberts
P.O. Box 8508
Asheville, North Carolina 28804

ANIMATION ON A STANDARD TERMINAL
by
Ralph Roberts

If your terminal runs fast (like mine at 9600 baud) it's easy to come up with some pretty interesting animation effects. You can develop some neat subroutines to add to games and so forth.

The short program in this article will print a cannon out on your video terminal and cause it to fire at a target. The cannon and target remain stationary but you see the muzzle flash and the shell's flight. This cannoneer never misses so the target will always explode.

Some of you hot programming types out there should be able to really come up with some great effects using this technique. I'm working on getting a little stick man to run across the screen myself.

This program is written in Smoke Signal Basic. The function of the CHR$ commands are as follows: CHR$(12) clears the screen, CHR$(07) is the bell, CHR$(08) is backspace, CHR$(26) spaces the cursor up one line.

If your terminal's speed is slower than mine, change the length of the wait commands. You may also change the speed of the cannon ball by doing the same thing.

Naturally, this won't work on hard copy -- just on CRT's. Try this little program next time you're just goofing around and see if it 'fires' your imagination.

```
*
0001 REM ::: CANNON ANIMATION PROGRAM :::
0002 REM :::::::::::by Ralph Roberts :::
0007 PRINT CHR$(12)
0008 LINE= 200
0009 PRINT TAB(32);"ARTILLERY TEST RANGE";P.TAB(36);"::: FIRING
0010 FOR X=1 TO 6:P.:NEXTX                                IN PROGRESS:::"
0011 FOR X=1 TO 76:P."-";:NEXT X
0015 FOR X=1 TO 5:P.:NDXT X
0020 PRINT "|::(}::::"
0022 PRINT "   OO    ";
0025 PRINT CHR$(26);
0028 PRINT TAB(70);"TARGET";:WAIT3
0029 PRINT :P.:P.:P.TAB(10);:FOR X=1 TO 3:P.CHR$(26);:NEXT X
0030 PRINT "** BOOM!! **";
0031 PRINT CHR$(07);
0032 PRINT " O ";
0034 WAIT .5
0036 FOR X=1 TO 15:PRINT CHR$(08);:NEXT X
0038 PRINT "                    O";
0039 WAIT .4
0040 FOR X=1 TO 5
0042 PRINT CHR$(08);" O";
0043 WAIT .4
0045 NEXT X
0050 PRINT CHR$(08);" !!!KAPOW!!!";
0051 FOR X=1 TO 5:P.CHR$(07);:NEXT X
0052 WAIT 1
0055 FOR X=1 TO 11
0056 PRINT CHR$(08);:NEXT X
0057 PRINT "          ";
0058 WAIT 5
0060 PRINT :P.
0062 PRINT TAB(10);
0070 PRINT CHR$(26);
0071 PRINT CHR$(26);
0075 GOTO 28
```

Mr. Don Williams, Editor
'68' MICRO JOURNAL
3018 Nagill Road
P. O. Box 849
Hixson, Tennessee 37343

Dear Don:

What an exciting pleasure to discover '68' MICRO JOURNAL. It seemed that most microcomputing publications were, for the most part, covering the more popular non-6800 systems. Of course, they must.

In the Nov./Dec. issue I found a most useful program submitted by Mr. Art Weller. While it was, I think, specifically written for TSC's Editor for use with MiniFlex, it extends Jim Thomas' concept of the 'DC' command. Thanks to both Art and Jim. I have applied Art's program to my Flex 1.0 DOS and TSC Editor and submit that specific application for other Flex 1.0 users.

I did run into a little problem. Since the TSC Editor likes the command table to be in alphabetical order. I chose to delete the 'NU' command to make room for the 'DC' command in the table. I first tried directly replacing the 'NU' command table entry with the 'DC' command table entry and received SYNTAX ERROR's for my trouble. It appears that the Editor searches the command table only through the commands which begin with the same letter of the alphabet as the typed command. This means I needed to rearrange the command table. I wrote a short program called JIFFLE (attached) which essentially shoves everything down five memory locations from $024E through $02B4. This destroys the contents of memory locations $02B5 through $02B9 -- i.e. the 'NU' command table entry. It also leaves a nice spot in the command table for the 'DC' entry at $024E through $0252.

With JIFFLE and a version of Art Weller's program for Flex 1.0 which I called EDPATCH (attached) here is all you do!

1. Patch Editor into memory using GET,EDIT.CMD
2. JIFFLE should be installed as a command. So the next step is to execute JIFFLE.
3. EDPATCH is a binary patch file and next step is thus: GET,EDPATCH
4. Finally, to save the whole mess: SAVE,NEWED,0020,1B71,0200
5. Install NEWED and you have your modified editor.

An sincere appreciation to Jim and Art, for they did the hard work. You can bet when I set up and going with FLEX 09 I'll make this modification to the T&C Editor also. Thanks for a fine magazine. After only receiving two issues I am very impressed with the sharing that is taking place and the quality of the material.

Sincerely yours,

John C. Tarvin

```
                    *   PROGRAM NAME: JIFFLE
                    *
                    *   EDIT COMMAND RELOCATOR
                    *
                    *   THIS PROGRAM ELIMINATES THE 'NU'
                    *   FORM OF THE NUMBERS COMMAND FROM
                    *   THE COMMAND TABLE OF TSC'S EDITOR
                    *   FOR 8" DMAFL, FLEX 1.0. IT ADJUSTS THE
                    *   COMMAND TABLE FOR INSERTING THE 'DC' COMMAND
                    *   AT THE LOCATION REQUIRED BY THE EDITOR'S
                    *   COMMAND ENTRY PROCESSOR.(I.E. ALPHABETICALLY)
                    *
                    *   IT ELIMINATES THE 'NU' COMMAND BY
                    *   SHIFTING THAT PORTION OF THE COMMAND
                    *   TABLE FROM THE 'O' FORM OF THE DELETE
                    *   COM AND TO THE 'NUMBERS' FORM OF THE
                    *   NUMBERS COMMAND DOWN 5 MEMORY LOCATIONS.
                    *   THIS LEAVES A 5 MEMORY LOCATION 'HOLE'
                    *   BETWEEN THE 'DELETE' AND 'D' FORM OF THE
                    *   OF THE DELETE COMMAND ($0 4E-$0252).
                    *
                    *   IT IS INTO THIS 'HOLE' THAT THE OVERLAY
                    *   'EDPATCH' PLACES THE NEW 'DC' COMMAND
                    *   INTO THE TABLE.
                    *
                    *   WRITTEN BY JOHN TARVIN 12/79
                    *
2000                        ORG     $2000
                    *
20 0 CE 02 B4   START   LDX     $02B4       LOWEST LOCATI N TO MOVE
2003 A6 00      LOOP    LDA A   0,X         FETC  IT
2005 A7 05              STA A   5,X         STORE IT 5 LOC FURTHER DOWN
2007 09                 DEX                 MOVE UP THE TABLE ONE LOCATION
2008 8C 02 4B           CPX     $024B       REACHED LAST ONE TO MOVE?
200B 26 F6              BNE     LOOP        IF NOT, MOVE ANOTHER
200D 7E AD 03           JMP     $AD03       IF SO, RETURN TO DOS
                        END     START

NO ERROR(S) DETECTED

                    *   PROGRAM NAME: EDPATCH
                    *
                    *   EDITOR-TO-DOS AND RETURN
                    *
                    *   FROM NOV./DEC. 19 9 '68' ARTICLE
                    *   BY ART WELLER. REFERENCE ORIGINAL
                    *   ARTICLE AUG. 1979 ISSUE '68' BY
                    *   JIM THOMAS.
                    *
                    *   APPLIED TO 8" TSC EDITOR, COPYRIGHTED
                    *   1978 USED WITH FLEX 1.0 ON DMAF1
                    *   APPLICATION BY JOHN TARVIN 12/79.
                    *
                    *   THIS EDITOR COMMAND PROVIDES A MEANS
                    *   OF EXECUTING DOS COMMANDS DIRECTLY FROM
                    *   THE EDITOR.  EDITOR FEEDS COMMAND TO THE
                    *   DOS COMMAND BUFFER AND TURNS CONTROL OVER
                    *   TO DOS.  WHEN THE COMMAND IS COMPLETED,
                    *   DOS RETURNS CONTROL TO THE EDITOR VIA
                    *   A 'WARM' RE-ENTRY.
                    *
                    *   CAUTION! THE 'NU' FORM OF THE 'NUMBERS'
                    *   COMMAND HAS BEEN DELETED FROM THE EDITOR
                    *   COMMAND TABLE TO ACCOMODATE THE 'DC'
                    *   COMMAND. TO USE THE 'NUMBERS' COMMAND
                    *   REQUIRES THAT THE ENTIRE COMMAND NAME BE
                    *   ENTERED.
                    *
                    *   SYNTAX: DC <dos command line>
                    *
                    *   EDITOR ADDRESS LOCATIONS
                    *
024E                APPADD  EQU     $024E       INSERT COMMAND IN TABLE
0615                SKIPSP  EQU     $0615       SKIP SPACES
0203                EDBAK   EQU     $0203       'WARM' EDITOR RE-ENTRY
035D                BEGPNT  EQU     $035D       BEGINNING DATA POINTER
00BD                BUFFER  EQU     $00BD       EDITOR INPUT BUFFER
0044                BUFFPT1 EQU     $0044       BUFFER LINE POINTER
                    *
                    *   FLEX 1.0 ADDRESS LOCATIONS
                    *
A080                DOSBUF  EQU     $A080       DOS LINE BUFFER
A0FF                DBEND   EQU     $A0FF       END DOS LINE BUFFER
AC14                BUFPNT  EQU     $AC14       DOS BUFFER POINTER
AD1E                PSTRNG  EQU     $AD1E       PRINT STRING
AD4B                DOCMND  EQU     $AD4B       EXECUTE DOS AS SUBROUTINE
                    *
                    *   MAKE ROOM FOR INIB ROUTINE
                    *
035D                        ORG     BEGPNT      CHANGE START OF EDIT FILE
035D 1B 72                  FDB     FINI        NEW START OF MEMORY
                    *
                    *   INSERT NEW COMMAND IN THE TABLE
                    *
024E                        ORG     APPADD      OVERLAY COMMAND TABLE
024E 44                     FCC     'DC' WITH NEW COMMAND
0250 00                     FCB     0
0251 1B 1E                  FDB     DC
                    *
                    *   START PATCH ROUTINE AT OLD
                    *   VALUE OF BEGPNT ($1B1E)
                    *
1B1E                        ORG     $1B1E       1ST AVAILABLE PLACE TO START
1B1E                DC      EQU     *           BEGIN ROUTINE
1B1E DE 44                  LDX     BUFFPT1     GET COMMAND LINE
1B20 BD 06 15               JSR     SKIPSP      SKIP LEADING SPACES
1B23 DF 44                  STX     BUFFPT1     SAVE COMMAND POINTER
1B25 CE A0 80               LDX     #DOSBUF     PUT START OF DOS BUFFER
1B28 FF AC 14               STX     BUFPNT      INTO DOS BUFFER POINTER
                    *
                    *   TRANSFER DOS COMMAND FROM EDITOR'S LINE
                    *   BUFFER TO DOS LINE BUFFER
                    *
1B2B FF 1B 56       DC1     STX     TEMPX2      SAVE DOS POINTER TEMPORARILY
1B2E DE 44                  LDX     BUFFPT1     POINT TO EDITOR BUFFER
1B30 A6 00                  LDA A   0,X         FETCH A CHARACTER
1B32 08                     INX                 BUMP EDITOR BUFFER POINTER
1B33 DF 44                  STX     BUFFPT1     SAVE IT
1B35 FE 1B 56               LDX     TEMPX2      GET DOS BUFFER POINTER
1B38 A7 00                  STA A   0,X         INSTALL THE CHARACTER
1B3A 08                     INX                 BUMP DOS BUFFER POINTER
```

```
  88  1D3A DC A0 FF          CPX   BMEND     HAVE WE OVERFLOWED BUS BUFFER?
  89  1D3E 27 0D             BEQ   ERROR     IF SO, EXIT WITH ERROR
  90  1D40 81 0D             CMP A #$D       ARE WE FINISHED?
  91  1D42 26 E7             BNE   BCL       IF NOT, FETCH NEXT CHARACTER
  92  1D44 BD A0 4B          JSR   DOCMND    IF SO, DO DOS COMMAND
  93  1D47 5D                TST B           WERE THERE ANY ERRORS?
  94  1D48 26 03             BNE   ERROR     IF SO, EXIT TO EDITOR WITH ERROR
  95  1D4A 7E 02 03          JMP   EDBAK     IF NOT, RETURN PEACEFULLY
  96
  97                  *   ERROR ROUTINE
  98
  99  1D4D            ERROR   EQU   *
 100  1D4D DE 1B 5B           LDX   #ERST     FETCH ERROR STRING ADDRESS
 101  1D50 BD A0 1E           JSR   PSTRNG    PRINT ERROR MESSAGE
 102  1D53 7E 02 03           JMP   EDBAK     RETURN AFTER THE BAD NEWS
 103
 104                  *   MEMORY RESERVATIONS
 105
 106  1D56            TEMPX2  RMB   2         TEMPORARY INDEX STORAGE
 107
 108                  *   MESSAGE STRING(S)
 109
 110  1D58 45         ERST    FCC   'ERROR IN DOS COMMAND LINE'
 111  1D71 04                 FCB   $04
 112
 113                  *   END OF ROUTINE, NOW BEGINNING OF
 114                  *   EDITOR DATA FILE
 115
 116  1D72            FINI    EQU   *
 117                  END

NO ERROR(S) DETECTED

TSC FLEX 1.0 EDITOR PATCH          12-2-79  TSC ASSEMBLER  PAGE    3


   SYMBOL TABLE:

APPADD 024E  BEGPNT 035D  BUFFER 0006  BUFFPT 0044  BUFPNT AD14
BBEND  A0F9  BC     1B1E  BCL    1B2B  BDCMND AD4B  BUSBUF A000
EDBAK  0203  ERROR  1D4D  ERST   1B5B  FINI   1D72  PSTRNG A01E
SKIPSP 0419  TEMPX2 1D56
```

Software Source Books™

Combining detailed descriptions with complete source listings, these books explain the internal operations and algorithms used in Hemenway Associate's popular systems software.

How much would such a complete software resource cost? If you've seen the PAPERBYTE books by Jack Hemenway and Robert Grappel you know how inexpensive they can be. And now you can have the companion volumes to the RA6800ML macro assembler and LINK68 linking loader books.

Remember, these are not just books; they are SoftwareSourceBooks ---- complete Software resources! Order them today; VISA and MasterCharge accepted.

CP/68™ OPERATING SYSTEM ($34.95)

* PIP Peripheral Interchange
* Program transfers data between physical devices
* Wildcard Filenames and Extensions
* Relocatable anywhere in Memory
* Extended Instr ction set includes 6809-type instructions (PSHX, PULX, etc)
* Device-independent I/O
* Random and Sequential Files
* Fits in less than 8K
* Chaining and overlaying
* Single Supervisor Call furnishes all DOS services
* Easily interfaced to new devices and peripherals
* Dynamic file allocation

STRUctured BAsic Language (STRUBAL+™) COMPILER for both business and scientific uses ($49.95)

* Variable precision from 4 to 14 digits
* Structured Programming forms
* Produces Relocatable and linkable code
* COMMON and DUMMY sections
* Extensibility
* String Handling
* Full scientific package
* Data structures with mixed data types

XA6809 Macro Linking Cross Assembler ($24.95)

* Runs on any M6800
* Full Macro facilities
* COMMON section for the production of ROMable code Conditional Assembly
* Generates linkable and relocatable code
* Sorted Symbol table listing
* Hash-coded Symbol table for speed

==========

Hemenway Associates Inc. 101 Tremont St. Boston MA 02108

Name	Title	Company

Street	City	State	Zip

( ) Check enclosed in the amount of $.....

( ) Bill VISA ( ) Bill MasterCharge

Card No..... Exp. Date.....

Please send the following books:

Add $0.75 per book to cover postage and handling

ATTENTION!

HOBBYISTS, EXECUTIVES, INVENTORS, ENGINEERS, PROGRAMMERS, SMALL BUSINESSMEN

LOOK AT WHAT IS AVAILABLE FROM STOCK
FROM THOMAS INSTRUMENTATION!

OUR NINTH YEAR IN BUSINESS

FEATURING

* * * * * NEW 16K (4-4K) MEMORY * * * * *

SS-50

*A "Cents"able way to add memory to your system
*Four 4K blocks individually addressable 0-F
*Additional memory at less than $10.00 per 1K
*Add memory 1K at a time, using low cost 2114's
*The Memory Card is available three ways

- Asm. & tested, socketed with all 16K **$295.00**
- Asm. & tested, socketed for 16K, with 1K **$129.00**
- Bare card and Documentation **$ 44.00**

SS-50

SPECIAL: $242.00 VALUE FOR ONLY $175.00

- A set of our bare cards to build a small system
- Consists of 8-Slot Backplane/Motherboard,
- Super CPU, Video Ram, 16K Memory, 10 Port
- Parallel I/O, Wire Wrap Prototype Card, and
- Documentation for each of the above

OUR SS-50 LINE-UP

- All Thomas Instrumentation's assembled cards are burned in at 150°F and fully tested
- All cards come with full documentation including software source listings where appropriate
- Bare card price does not include edge connectors
  - Super CPU asm. with monitor source but without 2K-2708's EPROM monitor **$195.00**
  - Monitor in two 2708's EPROMS **$ 29.00**
  - CPU bare card, doc., & source **$ 49.00**
  - Video ram asm. 7X9 char. 64X16 line **$169.00**
  - Video ram bare card, doc., & source **$ 45.00**
  - Parallel I/O asm. 100 I/O lines includes 5 PIA's for 10 ports **$110.00**
  - Parallel I/O bare card & doc. **$ 35.00**
  - Wire-Wrap/Prototype bare card **$ 29.00**

NEW PRODUCT LINES

- NEW RCA sealed Keyboards
  - Model 611 **$ 85.00**
  - Model 601 **$ 65.00**
  - CPU Cable **$ 10.00**
- Leedex Monitor **$139.00**

NEW BACKPLANES/MOTHERBOARDS

- The following cards are extra thick (3/32)
  - 16 Position SS-50 **$80.00**
  - 12 Position SS-50 **$60.00**
  - 8 Position SS-50 **$40.00**
  - 4 Position SS-50 **$20.00**
  - 8 Position SS-30 **$39.00**
- SS-50 to SS-30 Transition card will be available next month
- Connectors for the above cards are separate, SS-50 take 5 for each pos., SS-30 take 3 each backplanes take males, main cards take females
- Males Tin $0.40ea. Gold $1.60ea.
- Females Tin $0.50ea. Gold $1.60ea.

DEALERS FOR SWTPC, GIMIX, AND TSC

WE DESIGN HARDWARE

WE WRITE SOFTWARE

THOMAS INSTRUMENTATION SPECIALIZES IN HELPING YOU DEVELOP LOW COST SYSTEMS TO MEET YOUR INDIVIDUAL COMPUTING NEEDS . . . LET US AUTOMATE YOUR LABORATORY
WE HAVE SPECIAL SYSTEMS AND PRICES FOR SCHOOLS

THOMAS INSTRUMENTATION

168 EIGHTH STREET AVALON, N.J. 08202 (609) 967-4280

N.J. RES. INCLUDE 5% SALES TAX
CONTINENTAL U.S.A. INCLUDE $2.00 SHIPPING, CANADA $5.00, FOREIGN $10.00

VISA

master charge

FUTURE FEATURES on the FIFTY

The **FIFTY BUS** gives you compatibility and choices of Hardware and Software offerings by other manufacturers and Software houses such as:

• SWTP • TSC • Microware • Microworks

And now **GIMIX** presents our **SS50C 6809 CPU** card and systems.

The 6809 CPU card will be available in a standard version and our **6809 PLUS** version that is fully socketed to allow adding options at anytime.

+ A 6840 timer package that provides 3 independent 16 bit counters is included on all **6809 PLUS** cards.
+ A 9511 or 9512 Arithmetic Processors option with its own independent crystal that allows you to use 2, 3, or 4 MHz parts in any combination with the 6809 running at 1, 1.5, or 2 MHZ.
+ 1K of scratchpad RAM
+ A Time of Day Clock option with battery back-up. With this option you can also substitute 1K of CMOS RAM that will also be battery backed up.
+ User selectable processor speeds without having to change the crystal.

32K of PROM, ROM or RAM. Both versions have 4 sockets that can each hold from 1K to 8K parts. Single or multiple voltage parts can be used on the **PLUS** version. The standard version only allows the use of single voltage parts.

All on board devices and options can use extended addressing so that they will only respond to that page to which they are set.

The card is double buffered and allows versatility in the use of software and memory address control disciplines.

Please note that this card does **not** have an on board baud rate generator, and must be used in systems where baud rates (if needed) are provided elsewhere in the system.

And looking into our Crystal Ball we are hoping to ship by the end of 1980 our:

GIMIX DISC CONTROLLER CARD

Like all **GIMIX** products, it is designed for reliability. The board uses a phase lock loop data separator. It will use DMA and can control up to four 5" or four 8" single or double sided, single or double density drives.

We plan to have the latest generation of software available including: TSC's FLEX and forthcoming UNIFLEX, Microware's/Motorola's OS-9 and BASIC 09, and CSI's UCSD Pascal.

By the 4th Quarter of 1980 we should be delivering the

GIMIX HIGH RESOLUTION GRAPHICS CARD SET

It is capable of 512 x 512 resolution using 32K of Static RAM. It needs 8K of memory space on the bus, can use extended addressing, and can occupy 32K, if desired.

GHOSTable. On board software control registers allow the memory to disappear and reappear on the bus.

For Color, 3 of these sets can be used with RGB monitors.

Supplied with fast, compact and powerful software driver routines including an interface for TSC's BASIC. The software supports both vector and character operations, with control of virtually all board features.

It will also be available in 256 x 256 and 640 x 240 resolution versions.

And looking further into the Future---the 68000 should arrive in 1981

a **GIMIX 68000 CPU** card that will not be a whole new ball game---
just a new CPU card that is being designed for use with our present mainframe that has a 15-50 pin and 8-30 pin Motherboard, 25 amp power supply, Memory and other I.O. and video cards, and our forthcoming Disc Controller Card.

GIMIX INC. 1337 WEST 37th PLACE • CHICAGO, IL 60609 • (312) 927-5510 • TWX 910-221-4055

THE SOURCE

FOR UCSD PASCAL™ ON THE 6809

IMMEDIATE DELIVERY FOR SWTPC USERS, 8" or 5¼" DISKETTE
SMOKE SIGNAL BROADCASTING USERS, INQUIRE

CSI-1	Operating Systems, PASCAL Compiler, Screen Editor, Filer, Linker, Library, Setup, Binder, Interpreter, BIOS	**$250.00**
CSI-2	BASIC Compiler, YALOE, (Line-editor for hard-copy terminals), Patch, Disassembler, Calculator	**$100.00**
CSI-3	MACRO Assemblers for 6809, 6800 and other Microprocessors	**$100.00**
	ALL THREE DISKS and MANUAL (SYSTEM)	**$419.00**

FREE! UCSD PASCAL USER'S MANUAL PLUS SWTPC IMPLEMENTATION NOTES WITH PURCHASE OF CSI-1

68000 Coming, Summer, 1980
68000 Assembler, Spring, 1980

OEM and DEALER INQUIRIES INVITED

CSI CONTROL SYSTEMS, INC.
MICROSYSTEMS DIVISION

1317 CENTRAL AVE. KANSAS CITY, KANSAS 66102

CALL TOLL-FREE (800) 255-4411

Continental U.S.A. Only
(Kansas Residents call 913/371-6136)

master charge THE INTERBANK CARD

VISA

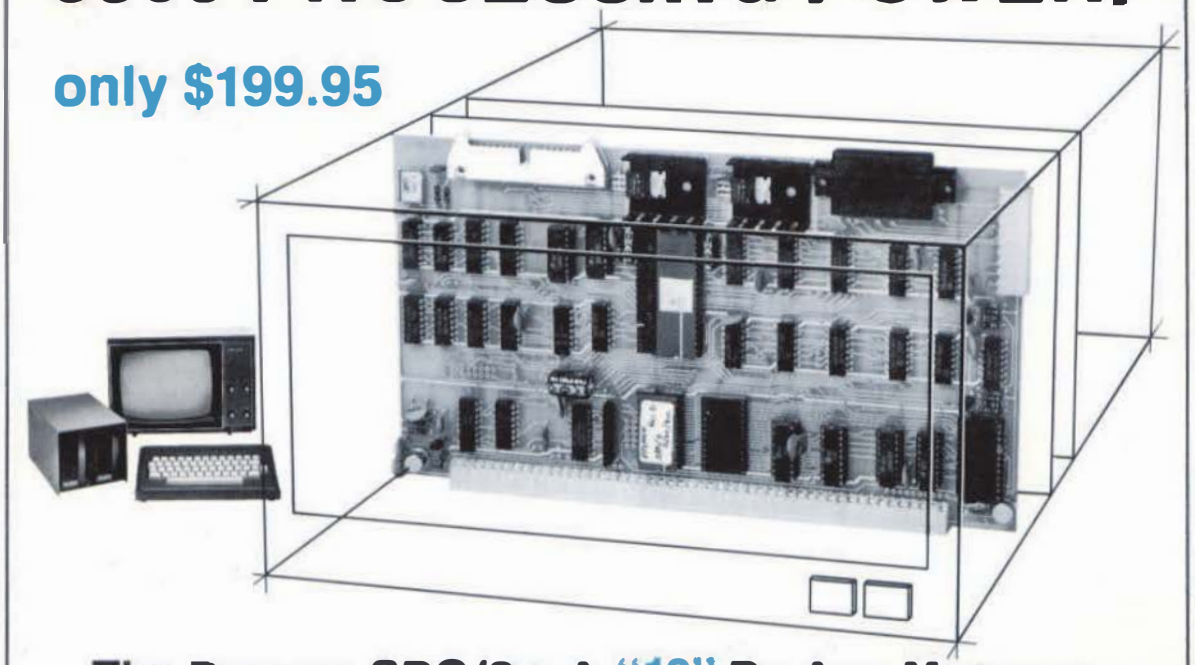
"UCSD Pascal" is a registered trademark of The Regents of The University of California

'68' Micro Journal
3018 Hamill Rd.
Hixson, TN 37343

Second Class Postage Paid
At Chattanooga, TN
ISSN 0194-5025

6809 PROCESSING POWER!

only $199.95



The Percom SBC/9™: A "10" By Any Measure.

Available with either the new, powerful 6809 μP or an optional 6800-software-compatible 6802, here are 10 beautiful reasons why the Percom SBC/9™ is not just another runner-up MPU/Single-Board-Computer card.

1. SS-50 bus direct, plug-in-compatible upgrade MPU. Requires **no** modification of the system bus, I/O or memory.
2. Full-capability stand-alone single-board computer. Accommodates a 6809 microprocessor or optional 6802 microprocessor **without modification.**
3. On-card 1 K ROM monitor "auto-links" to optional second 1 K PROM — if installed. Second PROM may be used to easily extend or modify the primary monitor command set.
4. Eight-bit parallel port is multi-address extension of system bus. Accommodates an exceptional variety of peripheral devices ranging from game paddles and keyboards to memory management modules. Connector is optional.
5. Serial port includes a full-range selectable bit rate generator. Optional subminiature 'D' connector provides RS-232 compatibility.
6. Extendable addressing via SS-50 bus baud lines to 1 Mbyte. Extendable addressing to 16 Mbytes or more through the parallel "super port."
7. Includes 1 Kbyte of static RAM.
8. All on-card I/O is fully decoded so that adjacent memory space may be used.
9. ROM circuit may be jumper-wired for single- or triple-voltage 2716 EPROM.
10. On-card power regulators simplify power supply design by minimizing regulation demands.

Plug the SBC/9™ into your SS-50 system bus, and just that easily you've upgraded to the new super-fast super-powerful 6809 MPU with such programming amenities as 10 addressing modes, 16-bit instructions, auto-increment/auto-decrement and position-independent code. Plus, you now have extended addressing capability, and operation under control of PSYMON™, the most powerful and flexibile 1K ROM 6809 operating system yet written.

Percom SYstem MONitor

PSYMON™ provides the usual ROM monitor functions in 1 Kbyte. It is easily extended and customized because its unique "look-ahead" program structure first searches an alternate command table. The table, if present, may be used to redefine or extend PSYMON's™ command set.

And with PSYMON™, I/O is easily directed to any peripheral device — even a disk system — through a Device Control Block table located in memory. This allows you to leave the details of I/O software to the separate I/O device drivers.

A PSYMON™ ROM is included free with the purchase of an SBC/9™. The Users Manual includes a source listing.

The 1 Kbyte ROM monitor for the SBC/9™ 6802 option includes a primary set of typical 6800-compatible monitor commands. As for PSYMON™, the commands are easily extended or modified.

Products are available at Percom dealers nationwide. **Call toll-free, 1-800-527-1592, for the address of your nearest dealer, or to order direct.** Prices and specifications subject to change without notice.

™ trademark of Percom Data Company, Inc.

PERCOM DATA COMPANY, INC.
211 N. KIRBY GARLAND, TEXAS 75042
(214) 272-3421

PERCOM